Universidad Carlos III de Madrid
Escuela Politécnica Superior

Departamento de Ingeniería Mecánica
Área de Ingeniería de Organización

EXPLORACIÓN DEL EMPLEO DE FÓRMULAS ELECTORALES EN ALGORITMOS GENÉTICOS

PROYECTO FIN DE CARRERA

Autor: Federico Egido Manso
Tutor: Miguel Gutiérrez Fernández

25 de mayo de 2010

"A esta conservación de las variaciones y diferencias individualmente favorables y la destrucción de las que son perjudiciales, la he llamado selección natural o supervivencia de los más aptos."

Charles Robert Darwin. Biólogo

Agradecimientos y dedicatoria

Este proyecto cierra una importante etapa de mi vida ya que supone el fin a varios años de carrera, no solo en lo que se refiere al 2º ciclo de la Ingeniería Industrial en la Universidad Carlos III de Madrid, sino también a la Ingeniería Técnica, el año de Erasmus en la TUG (Technische Universität Graz) y el Ciclo Formativo de Grado Superior en Automoción. Durante todos estos años de formación, compañeros, profesores, amigos y muchas otras personas, han estado a mi lado y me han ayudado a que me diera cuenta de lo que era capaz y lo que podría llegar a alcanzar, por lo cual les estoy muy agradecido.

Agradecimientos en especial a mi tutor, Miguel Gutiérrez Fernández, profesor del área de Ingeniería de Organización, por la paciencia y dedicación mostradas durante la realización, pero sobre todo por confiar en mí al escogerme para este proyecto tan emocionante.

Agradecimientos a mi familia por el apoyo mostrado, no solo económicamente, sino también emocionalmente en estos años de carrera. Las discrepancias, esfuerzos, sacrificios y decepciones han sido tanto para mí como para ellos enormemente recompensados por los buenos resultados.

Por último este proyecto se lo dedico a una persona muy especial que siempre ha estado a mi lado, me ha apoyado en cada una de las decisiones tomadas en estos últimos años y que ha hecho que sacara lo mejor de mí. Esta persona es en parte, partícipe de este proyecto y me ha ayudado enormemente a llevarlo a cabo, por lo que cualquier éxito o reconocimiento que se recoja, le pertenece también a ella.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este proyecto surge de la idea de explorar el uso de las fórmulas electorales en un campo que con el que a priori guarda poca relación: los algoritmos genéticos. En estas primeras páginas se realiza una introducción al trabajo desarrollado. Se presenta en primer lugar el origen y la motivación para llevarlo a cabo. Seguidamente, se plantean los objetivos que se desean alcanzar. A continuación se describe cómo se estructura todo el documento. Por último, se representa el cronograma que recoge la duración y la secuencia de las tareas realizadas.

1.1. Antecedentes y motivación

Los algoritmos genéticos forman parte de una familia denominada computación evolutiva (algoritmos evolutivos) [Mic96], una rama de la inteligencia artificial orientada a la resolución de problemas de optimización. Se fundamenta en la emulación del principio de evolución de las especies. En particular John Koza define los algoritmos genéticos como *"algoritmos de búsqueda probabilística que iterativamente transforman un conjunto (población) de objetos matemáticos individuales (normalmente cadenas binarias de longitud fija), cada uno de ellos asociado a un valor de salud (aptitud), en un nuevo conjunto de objetos (población de descendientes) usando el principio de selección natural de Darwin y una serie de operaciones genéticas naturales, como son el cruce (recombinación sexual) y la mutación"*, (adaptación del libro *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [Koz92]).

Imitando esta selección natural, los algoritmos genéticos parten de una población inicial compuesta de posibles soluciones al problema y operan sobre ella. Las tres operaciones básicas (o los tres operadores) que se realizan generación tras generación sobre esta población son la selección, el cruce y la mutación.

Desde sus orígenes, los algoritmos genéticos se han ido desarrollando e incorporando posibles variantes y nuevos operadores avanzados [Gol89a] [Hol75] [Mic96] que incrementan las ventajas de su uso en problemas de optimización.

Uno de los elementos que más relevancia tiene en el funcionamiento del algoritmo y al que se ha prestado mayor atención, es el operador selección. Este operador es el encargado de emular el principio de selección natural de manera que los mejores individuos (soluciones del problema) reciban mayores probabilidades de tener descendientes en la siguiente generación. En definitiva, se trata de un problema de reparto de un conjunto finito y discreto de descendientes a un conjunto finito y discreto de individuos. Este mecanismo de reparto puede hacer que un algoritmo genético obtenga muy buenos resultados o por el contrario muestre un comportamiento mediocre. Se han propuesto por este motivo gran número de variantes, a fin de poder escoger el más apropiado según las condiciones del problema a resolver.

Paralelamente a los algoritmos genéticos, en el mundo electoral existe la misma necesidad de repartir un conjunto finito y discreto, en este caso escaños electorales, a un conjunto finito y discreto de partidos políticos. Ya en los primeros compases de la historia, se recurría a distintos métodos de reparto de escaños, como en el caso de la *democracia ateniense* [Adr88], donde algunos cargos se elegían por votación y otros por sorteo. Actualmente, para realizar el reparto de escaños se recurre a las fórmulas electorales, que *"constituyen el mecanismo que se aplica para la distribución de los escaños y puestos electivos con base en los resultados de una votación"* [Rae71].

Vemos por tanto, que estos dos mundos comparten la necesidad de contar con un mecanismo (operador) de reparto. Diversas fórmulas utilizadas en el reparto de escaños electorales se han visto de algún modo reflejadas en técnicas heurísticas usadas en métodos de optimización. Se llega a dar el caso de que una de estas técnicas de reparto, la variante Hare, utilizada en el reparto de escaños electorales, coincide con el método de selección del *valor esperado* utilizado en los algoritmos genéticos (sección 3.3), lo que denota que el problema es común y que existe de hecho un paralelismo entre ambos mundos. La figura 1.1 muestra un esquema de la relación entre los algoritmos genéticos y el mundo electoral.

De entre las fórmulas electorales, las de divisores comunes (ver apartado 2.2.2.2) son las más empleadas en los sistemas de gobierno actuales. Cabe entonces la posibilidad de que también se puedan aplicar exitosamente como reparto de candidaturas a ser padre de los individuos de la siguiente generación, es decir como

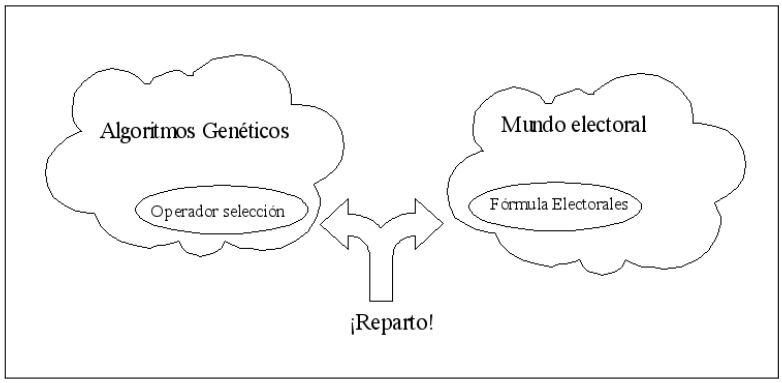

Figura 1.1: Relación entre los algoritmos genéticos y el mundo electoral

mecanismo de selección de los algoritmos genéticos. Hasta donde hemos podido comprobar, ninguna fórmula de divisores comunes se emplea como base en el operador selección de los algoritmos genéticos. No están presentes como opción disponible en los software de optimización de algoritmos genéticos de uso más extendido. Por lo tanto, se presenta una atractivo campo de investigación consistente en el análisis de qué resultados se obtendrían si estas fórmulas se aplicasen en los algoritmos genéticos.

1.2. Objetivos

Acabamos de ver en los antecedentes del proyecto que existe un gran campo a estudiar basado en el empleo de las fórmulas de divisores comunes en los algoritmos genéticos. Este es precisamente el planteamiento genérico de este proyecto, que consiste en explorar cómo funcionaría un algoritmo genético con un mecanismo de selección que utilice alguna de estas fórmulas frente a un algoritmo genético que utilice los operadores de selección habituales.

Como se ha comentado, en muchos programas y en la bibliografía se presentan diversas variantes u opciones habituales del operador selección. Esto también ocurre con el resto de operadores, ya que, dependiendo del problema a optimizar, unos dan mejores resultados que otros. Basándonos en esto, para llevar a cabo el planteamiento genérico, tendríamos que recurrir a comparar el comportamiento de diversos algoritmos genéticos basados en los diferentes operadores de selección habituales, con el comportamiento del conjunto de algoritmos genéticos que incorporen las fórmulas electorales de divisores comunes más empleadas, frente a una batería completa de problemas de optimización. Para mayor complejidad existe una dependencia del comportamiento de un operador con la elección del

resto de operadores (en este caso, cruce, mutación y operadores adicionales) de entre las variantes respectivas posibles.

Por limitaciones de alcance del trabajo, se plantea desde el inicio el carácter prospectivo del análisis a realizar, de manera que se ve necesario acotar las dimensiones de la comparación. Se busca que esta acotación no impida obtener resultados representativos. Se decide definir el siguiente entorno:

- Algoritmo Genético Simple (AGS). De entre la infinidad de variantes de algoritmos genéticos, se limita el estudio al algoritmo genético simple. Goldberg, uno de los autores que son referencia indiscutible en la génesis de estos algoritmos, lo presenta como punto de partida y referencia para todas las variantes que se han desarrollado [Gol89a]. Pese a su simplicidad, ofrece buenos resultados y es empleado frecuentemente como base para la comparación en estudios [Coe95]. El algoritmo genético simple utiliza la representación binaria del individuo, determina una población inicial al azar, mantiene constante el tamaño de la población generación a generación y recurre exclusivamente a los operadores selección, cruce simple (o por un punto) y mutación con tasa constante, para buscar el óptimo. Es característico sobre todo por el denominado método de selección de la ruleta, en el cual, a cada individuo de la población se le asigna una probabilidad de ser padre de un individuo de la siguiente generación, proporcional al valor de salud o aptitud (referenciado al valor de la función a optimizar) que tiene en relación con la salud total de la población. Se identifica el valor de la salud relativa con una porción de una ruleta ficticia, y se determina el número de hijos que le corresponde a cada individuo haciendo girar sucesivamente esta ruleta ficticia mediante una secuencia de números aleatorios.

- Un problema de optimización. Se plantea la exploración sobre un único problema de optimización, que sea particularmente representativo y comúnmente utilizado en otros estudios y experimentos. La consulta a la bibliografía permite constatar que existe un conjunto de problemas de optimización enfocados a la experimentación, los cuales reciben comúnmente el nombre de función test.

- Una fórmula electoral. De entre todas las fórmulas electorales de divisores comunes, buscaremos aquélla que, siendo particularmente representativa, ofrezca un mayor atractivo y potencial para su aprovechamiento en la dinámica de los algoritmos genéticos.

- Un procedimiento de evaluación. Para comparar el rendimiento de dos algoritmos genéticos, existe una variedad de parámetros y criterios establecidos

en la literatura (procedimientos de evaluación). Buscando que no se altere en ningún sentido las conclusiones del análisis, se plantea encontrar un procedimiento existente y reconocido en lugar de establecer un conjunto propio de parámetros de comparación.

Partiendo del planteamiento genérico y con las consideraciones anteriores, establecemos el objetivo principal como:

> OBJETIVO PRINCIPAL *Comparar el comportamiento del algoritmo genético simple -con el operador Ruleta- en la resolución de un problema de optimización representativo, frente a una variante del algoritmo genético simple que incorpore un nuevo operador selección basado en alguna de las fórmulas electorales de divisores comunes.*

Atendiendo a esta necesidad de comparación, surgen varios objetivos instrumentales. Estos objetivos constituyen hitos intermedios a alcanzar que propicien la consecución del objetivo principal.

> OBJETIVO INSTRUMENTAL 1 *Determinar cuál de las fórmulas electorales de divisores comunes más conocidas resulta la más atractiva como operador selección en algoritmos genéticos.*

> OBJETIVO INSTRUMENTAL 2 *Determinar una batería de problemas de optimización utilizados frecuentemente en estudios y experimentos de algoritmos numéricos de optimización, y seleccionar el problema de optimización a utilizar.*

> OBJETIVO INSTRUMENTAL 3 *Identificar los procedimientos de evaluación empleados en estudios y experimentos de algoritmos de optimización y definir el procedimiento a emplear.*

Adicionalmente, para poder llevar a cabo el estudio, surge el siguiente objetivo instrumental:

> OBJETIVO INSTRUMENTAL 4 *Encontrar un software de computación evolutiva que permita la ejecución del algoritmo genético simple frente a un problema de optimización a definir y que tenga la opción de código accesible para poder programar y utilizar el nuevo operador selección.*

1.3. Estructura del documento

El documento recorre el proceso seguido para satisfacer el objetivo principal y los objetivos instrumentales del proyecto, consistente esencialmente en una etapa previa de selección de software de computación evolutiva, fórmulas electorales, procedimientos de evaluación y baterías de funciones test, seguida de la programación del operador, y por último de la experimentación y la obtención de las conclusiones. Más concretamente, tras este capítulo de introducción, la estructura del documento se desarrolla como sigue:

Capítulo 2: marco teórico Se realiza una breve introducción teórica a los algoritmos genéticos y a las fórmulas electorales. El objetivo es, primeramente, familiarizar al lector con estos algoritmos destinados a la optimización y basados en los mecanismos de selección de la naturaleza (algoritmos genéticos). En segundo lugar, se describen las fórmulas electorales más comunes que se pueden encontrar en la bibliografía política, utilizadas con mayor o menor frecuencia en el reparto de escaños.

Capítulo 3: alternativas y solución tomada Se desarrollaran los objetivos instrumentales 1, 2, 3 y 4. Para ello, primeramente se recoge una recopilación de software de computación evolutiva, baterías de funciones test y procedimientos de evaluación. A continuación se justifica la selección del software, la batería de funciones, el procedimiento de evaluación y la función objetivo así como de la fórmula electoral de divisores comunes que posteriormente se utilizarán para realizar la experimentación.

Capítulo 4: desarrollo Contiene el desarrollo del software de algoritmos genéticos *GA toolbox* y la creación de un nuevo programa para la recopilación de los datos, *Statistics*, en base a los criterios del procedimiento de evaluación. Previamente se presenta una descripción de GA toolbox, software de algoritmos genéticos, con el fin de conocer el funcionamiento, las opciones y los ficheros que componen este software.

Capítulo 5: experimentación Se define en primer lugar cómo se va a realizar el experimento, las estrategias adoptadas y, a continuación, se muestran los resultados más relevantes de la experimentación, objetivo principal del proyecto.

Capítulo 6: conclusiones y trabajo futuro Recoge el análisis de las conclusiones obtenidas de la realización del proyecto y plantea las líneas de continuación

del análisis exploratorio.

Por último, se han incluido los siguientes apéndices con el fin de complementar los contenidos de los capítulos que recogen las actividades principales del proyecto.

Apéndice A: Métodos multicriterio aplicados Se muestran los resultados de la aplicación de los 3 métodos multicriterio para la selección del software de computación evolutiva.

Apéndice B: aplicación fórmulas distributivas Se facilita la comprensión de las fórmulas distributivas explicadas en el apartado 2.2.2 por medio de una hipotética aplicación de las variantes de fórmula distributiva a un mismo supuesto, tomada de la bibliografía [CG97].

Apéndice C: GA toolbox En este apéndice se incluye el informe técnico original del software GA toolbox [Sas07].

Apéndice D: funciones test y procedimientos de evaluación Se recogen las alternativas de baterías de funciones test y procedimientos de evaluación, que se describen en el capítulo 3. Además, se describen las características y aspectos más importantes que deben recoger estas alternativas así como los problemas que pueden presentar.

Apéndice E: ejemplo batería de funciones Se presenta una selección de 10 funciones de entre las que componen el grupo de 25 funciones del criterio de evaluación escogido, realizada con la intención de tener un grupo reducido de funciones lo más completo y homogéneo posible.

Apéndice F: resultados obtenidos Se muestra una relación de los resultados obtenidos en la experimentación. Además se incluyen algunas gráficas adicionales como complemento a estos criterios.

1.4. Cronograma

Una vez definidos los objetivos y descrito cómo se va a estructurar el documento, se incluye en este capítulo un diagrama de Gantt o cronograma de tareas seguido.

El desarrollo de este proyecto se ha simultaneado con la actividad profesional, lo que ha condicionado la dedicación semanal prestada, además de alargar la duración de las tareas. La duración total ha sido de unos 25 meses. Se ha mantenido una continuidad en la dedicación, aunque alternando periodos de mayor o menor actividad. Se estima que en el global de la duración del proyecto la dedicación semanal promedio ha sido de 16 horas/semana, lo que supone una dedicación del 40 %, tomando como referencia (100 %) una jornada estándar de 8 horas/día, 40 horas/semana.

La figura 1.2 muestra el diagrama de Gantt del proyecto. Este cronograma puede utilizarse para tener una estimación del coste en horas/hombre de las distintas tareas del proyecto. Como se puede ver, se ha llevado una ejecución escalonada de las tareas a excepción de la redacción de la memoria que se ha ido avanzando en mayor o menor medida según se completaban las tareas correspondientes. El tiempo transcurrido desde la finalización hasta la presentación se ha dedicado a la preparación de la lectura.

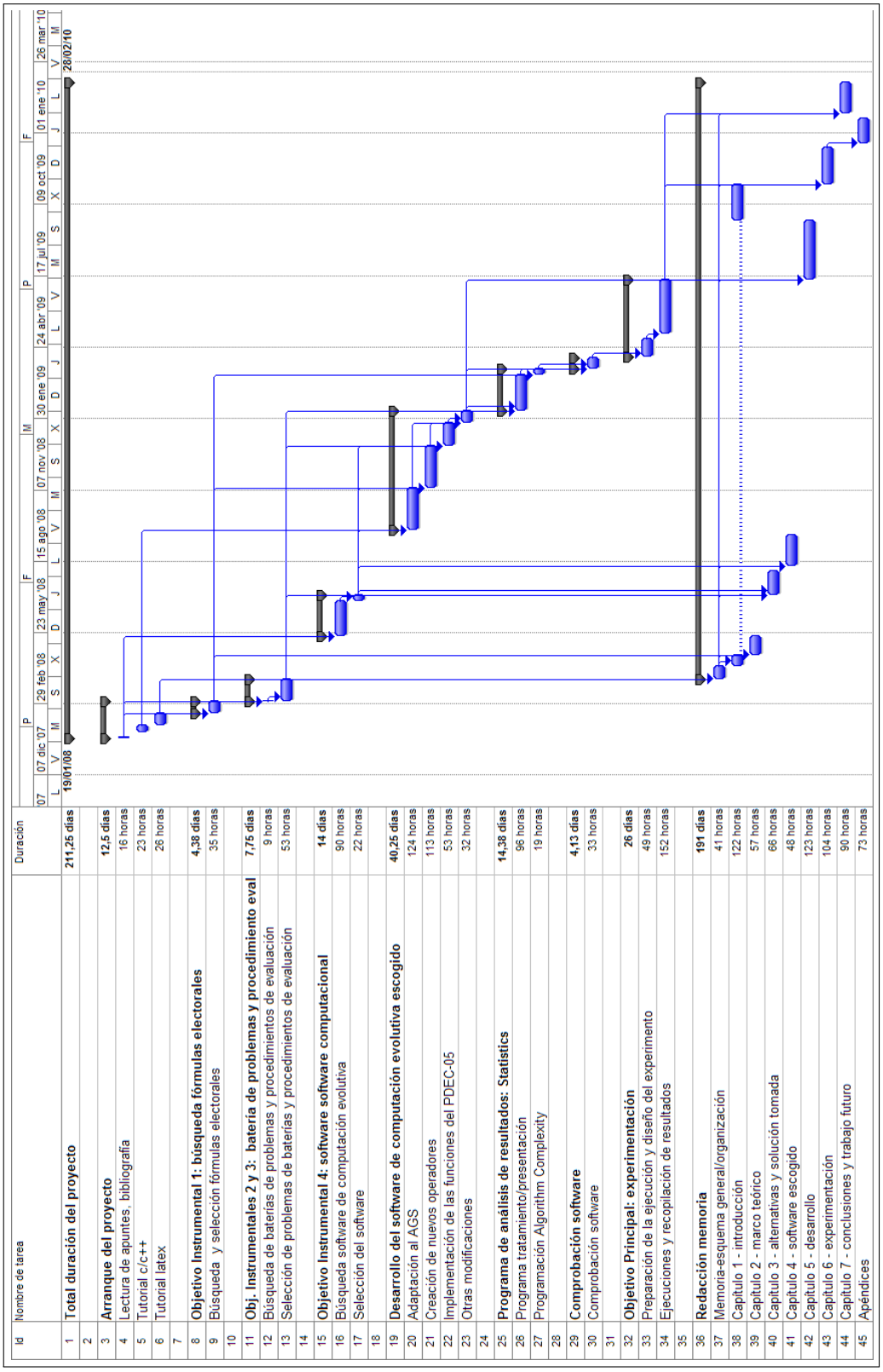

Figura 1.2: Cronograma del proyecto. Diagrama de Gantt

Capítulo 2

Marco teórico

Este capítulo consiste en una breve introducción a los aspectos teóricos del proyecto con el fin de familiarizar al lector con los algoritmos genéticos y las fórmulas electorales.

Primeramente se tratarán los algoritmos genéticos en un sentido amplio y en particular el algoritmo genético simple. En segundo lugar se describen las fórmulas electorales más comunes que se pueden encontrar en la bibliografía política, utilizadas con mayor o menor frecuencia en el reparto de escaños.

2.1. Introducción a los algoritmos genéticos

En esta sección se muestra una breve introducción a los algoritmos genéticos, extraída en su mayoría del libro del autor David Edward Goldberg, "Genetic Algorithms in Search, Optimization & Machine Learning" [Gol89a], uno de los autores de referencia en la materia.

Este texto, es una de las principales referencias sobre algoritmos genéticos y del que seguramente muchos de los expertos hayan tenido o tengan un ejemplar en sus manos.

2.1.1. Definición

En la introducción (capítulo 1) figura la siguiente definición de John Koza de algoritmos genéticos: *"algoritmos de búsqueda probabilística que iterativamente transforman un conjunto (población) de objetos matemáticos individuales (normalmente cadenas binarias de longitud fija), cada uno de ellos asociado a un valor de salud (aptitud), en un nuevo conjunto de objetos (población de*

descendientes) usando el principio de selección natural de Darwin y una serie de operaciones genéticas naturales, como son el cruce (recombinación sexual) y la mutación", (adaptación del libro *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [Koz92]).

De esta definición se deduce que:

- Es un método para la resolución de problemas de optimización.
- Es un método iterativo que rastrea el área de búsqueda por medio de una población de individuos (cadenas) que representan conjuntos de soluciones.
- Están basados en la selección natural: herencia genética y principio de supervivencia.
- Están concebidos y diseñados para su implementación y ejecución en computador.

Los algoritmos genéticos forman parte de una familia denominada computación evolutiva (o algoritmos evolutivos), que incluye además las estrategias de evolución, la programación evolutiva y la programación genética [Mic96].

2.1.2. Elementos

Imitando la selección natural, los algoritmos genéticos operan sobre una población compuesta de posibles soluciones al problema, tal como se mencionó en la introducción. Para poder llevar a cabo esto, será necesario definir una representación del problema y una serie de operaciones (operadores).

2.1.2.1. Representación: Fenotipo y Genotipo

Para proceder a la representación de cada posible solución al problema (*fenotipo*), es necesario recurrir a una codificación, generalmente binaria, en una o varias cadenas que se denominan "cromosomas" (*genotipo*). Cada individuo de la población es el representante, dentro del algoritmo genético, de una posible solución al problema y puede, por lo tanto, estar formado por uno o varios cromosomas. Este "paralelismo" entre el genotipo y fenotipo se detalla gráficamente en la siguiente figura (2.1).

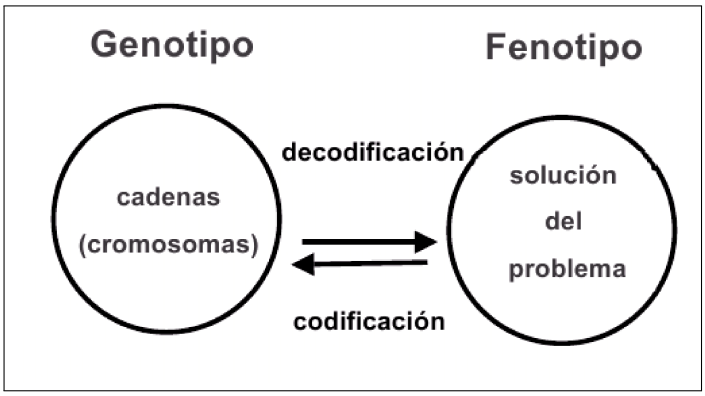

Figura 2.1: Fenotipo y genotipo

Como se puede ver en la figura 2.2, el cromosoma se encuentra dividido en subcadenas que reciben el nombre de genes. A cada posible valor del gen se le denomina alelo. Dependiendo de la bibliografía o el problema, el gen puede representar el valor de una variable del problema o el elemento más pequeño de la cadena (el bit: 0/1).

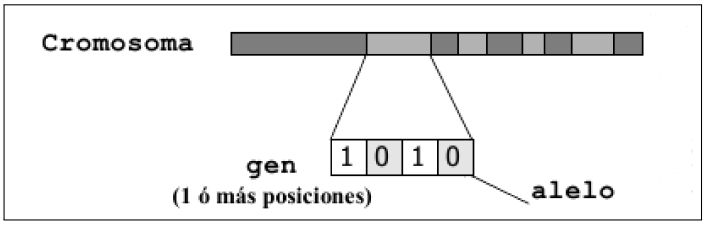

Figura 2.2: Representación de un cromosoma con codificación binaria y un gen con 4 bits [Nes07]

2.1.2.2. Operadores

Los operadores son las operaciones que se utilizan para formar las siguientes generaciones (descendencia). Los operadores básicos son [RP02]:

- Selección: este operador escoge individuos entre la población para efectuar la reproducción. Cuanto más capaz sea el individuo, más veces será seleccionado para reproducirse. Objetivo: mantener las características de aquellos individuos mejor adaptados.
- Cruce: se trata de un operador cuya labor es intercambiar secuencias (o porciones de cadena) entre dos individuos, para crear nueva descendencia.

Imita el intercambio de cadenas de dos cromosomas. Objetivo: garantizar explotación de buenas secciones del espacio de búsqueda.

- Mutación: este operador produce variaciones de modo aleatorio en los cromosomas de los individuos. La mutación podría darse, por ejemplo, en un bit de una cadena, con una probabilidad, normalmente muy pequeña. Objetivo: introducir diversidad (aleatoriamente) y explorar nuevas secciones del espacio de búsqueda [1].

2.1.3. Operador selección

Dado que el planteamiento genérico del proyecto está enfocado, dentro de los algoritmos genéticos, en el operador selección, se realiza en este apartado una descripción más amplia de este.

Como se ha comentado, el operador selección es el encargado de escoger los individuos de la población que van participar en el cruce y la mutación, con el fin de transmitir y conservar aquellas características de la soluciones que se consideran valiosas a lo largo de las generaciones. Para ello, aquellos individuos mejor adaptados, a través de su valor de la función de salud, tendrán más probabilidades de reproducirse. Sin embargo, es necesario también incluir un factor aleatorio que permita incluir a individuos que, aunque no estén muy bien adaptados, puedan contener alguna información útil para posteriores generaciones, con el objeto de mantener así también una cierta diversidad en cada población.

Aunque existen múltiples operadores selección, a continuación se presentan varias alternativas de selección que comúnmente se pueden encontrar en la bibliografía especializada. En este caso, estos operadores han sido extraídos de [Gol89a].

1. Ruleta: con este método la probabilidad que tiene un individuo de reproducirse es proporcional a su valor de función de evaluación, es decir, a su adaptación. El método consiste en asignar cada porción de una ruleta ficticia acorde a la probabilidad obtenida para cada individuo. Se determina el número de hijos que le corresponde a cada individuo haciendo girar sucesivamente esta ruleta ficticia mediante una secuencia de números aleatorios. Dado que forma parte del algoritmo genético simple, se realiza una descripción más detallada en siguiente apartado 2.1.5.

2. Selección por torneo: se selecciona un grupo de t individuos (normalmente t = 2, torneo binario) y se genera un número aleatorio entre 0 y 1. Si este

[1] Con ello se persigue impedir que el algoritmo quede atrapado en óptimos locales, creando un "salto" en la estructura, que en ocasiones resulta próximo a un óptimo mejor o incluso al global

número es menor que un cierto umbral K (usualmente 0,75), se escoge para la reproducción al individuo con mejor adaptación. Si por el contrario este número es menor que K, el individuo con peor adaptación es el que pasaría a las siguientes fases (cruce y mutación).

3. Valor esperado: este reparto se realiza en dos fases. Primeramente se calcula la probabilidad de selección de cada individuo tomando su valor de salud normalizado y se multiplica por el número de descendientes esperado. Seguidamente se reparten el nº de hijos con arreglo a la parte entera de estos cocientes (1ª fase). Con ello no se habrá podido alcanzar el número de individuos necesario (conjunto de padres). Para para completarlo se recurre al resto de los cocientes con el fin de repartir entre los candidatos el número de descendientes que queda pendientes (2ª fase). Se utilizan las siguientes técnicas:

   a) Reparto determinista: los aspirantes se ordenan en una lista según el valor del resto, de mayor a menor. Entonces, la descendencia pendiente se va asignando por orden de lista.

   b) Reparto estocastico con remplazamiento: es el resultado de aplicar la ruleta al valor esperado. Los restos son usados para calcular las probabilidades que compondrán la ruleta (porciones). De igual modo, por medio de números aleatorios, se va asignando el número de hijos que le corresponde a cada candidato hasta completar la desdendencia pendiente.

   c) Reparto estocastico sin remplazamiento: esta técnica toma los restos de los individuos aspirantes como probabilidades de asignación. Entoces, uno a uno, se obtiene un número al azar y se comprueba si esta dentro de la probabilidad. En tal caso se asigna a este candidato a tener descendencia (se le asigna un hijo). El proceso continua hasta que se se agota el número de descendientes pendientes.

Como se puede ver, el factor aleatorio a utilizar es distinto en cada uno. Por ejemplo, para la ruleta se realiza una selección es aleatoria según su valor de salud, mientras en el ranking se recurre al ordenamiento de la población. Estas diferencias conllevan resultados distintos según se escoja un operador u otro. Por lo tanto, tal como se ha comentado en la introducción, el operador selección es uno de los elementos que más relevancia tiene en el funcionamiento del algoritmo.

2.1.4. Descripción de funcionamiento

La figura 2.3 representa el esquema de funcionamiento del algoritmo genético, el cual se detalla en este apartado.

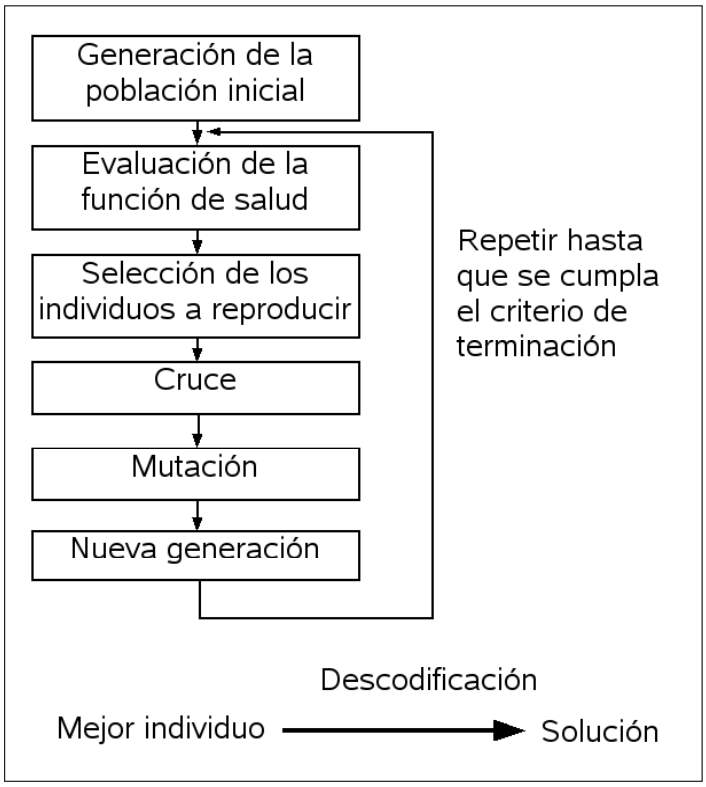

Figura 2.3: Esquema de funcionamiento del algoritmo genético (adaptado de [YBS+03])

Este proceso [YBS+03] comienza seleccionando un número de individuos para que conformen la población inicial.

A continuación se evalúa la función de salud para estos individuos (en lo sucesivo denominaremos a la evaluación de la función objetivo FES) para obtener el valor de salud. La función de salud da una medida de la aptitud del individuo para sobrevivir en su entorno (por decirlo de otro modo, su fortaleza). Debe estar definida de tal forma que los individuos que representen mejores soluciones tengan valores más altos de adaptación.

Los individuos más “fuertes”, es decir, los que mayor valor de salud presentan, se seleccionan en parejas para reproducirse por medio del cruce. El cruce genera

nuevos individuos que combinan características de ambos padres. Estos nuevos individuos reemplazan a los individuos con menores valores de salud.

A continuación, algunos individuos son seleccionados al azar para ser mutados. La mutación consiste en aplicar un cambio aleatorio en su estructura (se permutan algún o algunos valores de bits).

El ciclo de selección, cruce y mutación se repite hasta que se cumple el criterio de terminación del algoritmo, momento en el cual el mejor individuo de todas las generaciones se devuelve como solución.

2.1.5. El algoritmo genético simple

Goldberg propuso por primera vez en su libro [Gol89a] el algoritmo genético simple, en lo sucesivo AGS. Como ya se habló en la introducción, los mecanismos del AGS son sorprendentemente simples, implicando nada más complejo que copiar cadenas e intercambiar secuencias. Sin embargo, este proceso ofrece buenos resultados, basándose en su simplicidad y efectividad.

Problema de optimización Con la finalidad de facilitar el entendimiento de los algoritmos genéticos y en particular el AGS, se ha incluido en este apartado el siguiente caso práctico a modo de ejemplo.

El problema propuesto consiste en maximizar la función 2.1 en el intervalo [0,31].

$$f(x) = x^2; \qquad x \in [0, 31] \tag{2.1}$$

2.1.5.1. Características del algoritmo genético simple

A continuación se describen, los operadores y técnicas usados como referencia para el AGS:

Representación Será en binario y a cada individuo solamente le corresponde un cromosoma (figura 2.4). La longitud de la representación depende de las características del problema (número de variables, número de funciones objetivo, dimensión del dominio) y de las características de la solución buscada (precisión deseada, por ejemplo).

Para nuestro problema, Goldberg utiliza un símil que nos ayuda a entender la codificación de las soluciones. Se trata de posicionar los 5 interruptores de un

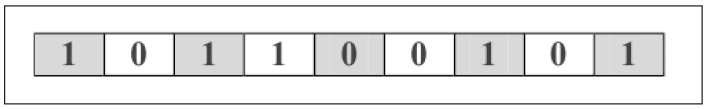

Figura 2.4: Representación cadena binaria: cromosoma [Nes07]

aparato (caja negra, ver figura 2.5) con el fin de obtener la mayor salida, es decir el óptimo. Para cualquier combinación de la posición de los interruptores existe una salida que viene determinada por la función $f = f(s)$, donde s es la posición de los interruptores. La función escogida para nuestro ejemplo será obviamente la función 2.1, cuyos valores de s se encontrarán entre [0,31].

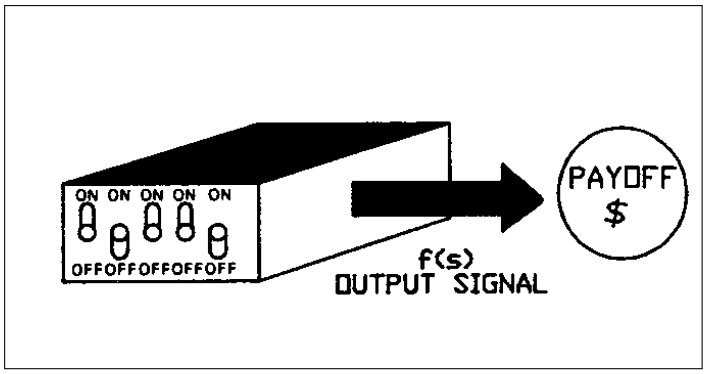

Figura 2.5: Representación cadena binaria: cromosoma [Gol89a]

El motivo de por qué son 5 interruptores y no más ni menos, es que un binario de 5 bits, permite 32 combinaciones posibles ($2^5 = 32$), lo que hace que sus valores pasados a decimal oscilen entre 0 y 31 precisamente. Con ello se determina, para nuestro problema, el tamaño del cromosoma en 5 bits con una codificación en binario y un solo cromosoma para cada individuo.

Población inicial Cada individuo que se toma para comenzar el algoritmo vendrá determinado al azar. Todos ellos formarán un conjunto, cuyo parámetro a decidir será N (número de individuos). La siguiente tabla 2.1 muestra el ejemplo una población al azar de $N = 4$ (4 individuos) para el problema propuesto:

Evaluación La evaluación de los individuos vendrá como resultado de la función de salud, que tratará de medir su aptitud. Caso de que sea posible, se puede usar la función objetivo como función de salud.

Individuo	Cromosoma	Valor (s)
1	01101	13
2	11000	24
3	01000	8
4	10011	9

Tabla 2.1: Población inicial [Gol89a]

La tabla 2.2 muestra el valor de salud de la población inicial de la tabla anterior (2.1). Como se puede comprobar la función de salud escogida es la del propio problema de maximización (función 2.1).

Selección Dado que escoger directamente a los individuos que han obtenido mayor valor de salud en la evaluación no refleja exactamente el mundo natural (en el cual, por ejemplo alguno de los mejores individuos muere por enfermedad) y por otro lado podría dar una convergencia prematura a un óptimo local, Goldberg escogió un método distinto para usarlo como operador de selección del AGS. Este método es la "ruleta", en el cual, a cada individuo de la población se le asigna una parte de la ruleta acorde a la proporción de su valor de salud frente a las demás (figura 2.6).

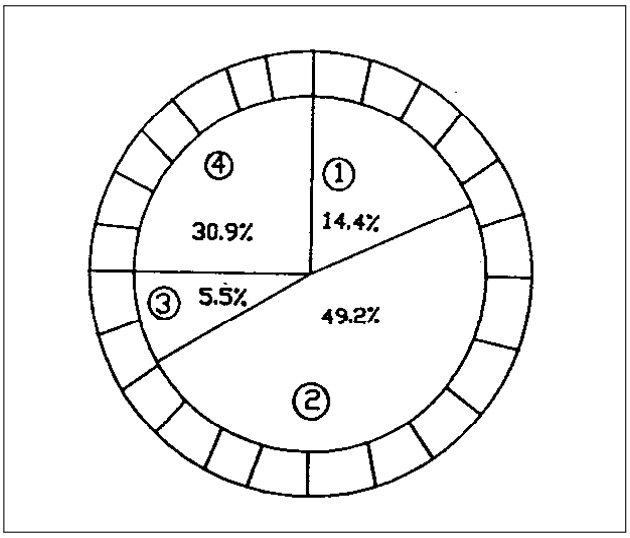

Figura 2.6: ruleta de selección [Gol89a]

Para nuestra población inicial, estos valores son los mostrados en la columna "% del Total" de la tabla 2.2. Así por ejemplo, al individuo 1 le corresponderían los valores 0-0.143 de la ruleta, al 2 los valores 0-144-0.491 y así sucesivamente hasta el último individuo.

Individuo	Cromosoma	Valor (*s*)	V. salud	% del Total
1	01101	13	169	14.4
2	11000	24	576	49.2
3	01000	8	64	5.5
4	10011	19	361	30.9
TOTAL			1170	100.0

Tabla 2.2: Ejemplo de población y sus valores de salud [Gol89a]

Para obtener el conjunto de padres de la siguiente generación, de entre todos los candidatos a descendencia, bastará con hacer girar la ruleta, tantas veces como individuos componen la población. Así, en cada giro, el resultado obtenido de la ruleta nos indicará el padre, según caiga en el rango perteneciente a ese individuo. Es posible, por lo tanto que un individuo pueda ser elegido padre para la siguiente generación varias veces, por lo que formará parte ese número de veces del conjunto.

Cruce El cruce consta de tres fases: emparejamiento, probabilidad de cruce (se determina si hay cruce) y cruce simple.

El emparejamiento se deberá realizar de manera aleatoria escogiendo de 2 en 2 a los padres del conjunto. Así, teniendo individuos que han sido elegidos padres varias veces, cabe la posibilidad que estos se emparejen consigo mismo. Como el modo en que se obtienen los padres de la siguiente generación por el método de la ruleta es aleatorio, no será necesario hacer un nuevo ordenamiento aleatorio de este conjunto de padres, por lo que se podrán emparejar según vayan saliendo. La tabla 2.3 muestra un posible resultado de la ruleta y emparejamiento sobre nuestra población inicial.

En el AGS el cruce que se aplica es el "cruce simple" con probabilidad 1 ($Pc = 1$). Primeramente se determina aleatoriamente (según la Pc) para cada pareja de individuos si se va a cruzar o pasaría a la siguiente generación tal cual. Para el AGS siempre se cruzan, ya que $Pc = 1$. A continuación se realiza el cruce simple, en el que se selecciona aleatoriamente una posición del cromosoma, a partir del cual los

Resultado de ruleta	Padres	Emparejamiento
0,020	1	
0,382	2	1-2
0,889	4	
0,524	2	4-2

Tabla 2.3: Ejemplo de selección y emparejamiento

dos padres intercambian sus valores. Por lo tanto se crea así la nueva descendencia. Por ejemplo consideremos que ha resultado el punto de cruce $k = 4$ (figura 2.7) para los individuos 1 y 2 de nuestro ejemplo (cromosomas A_1 y A_2):

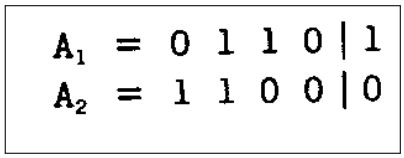

Figura 2.7: Punto de cruce [Gol89a]

Según lo explicado, se intercambiaría el bit 5 de los dos individuos, quedando como descendencia A'_1 y A'_1 de la figura 2.8.

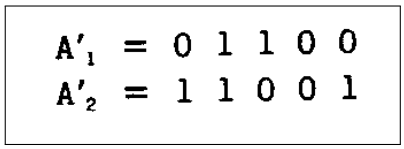

Figura 2.8: Descendencia [Gol89a]

Mutación Esta será para el AGS con tasa constante, normalmente muy pequeña (en torno al 0,05 %), y aplicada a cada bit del cromosoma. Si por probabilidad, un bit es elegido para mutar, se permuta el valor de este (1 ó 0). Cualquier individuo de la población puede resultar mutado e incluso puede ocurrir que varios bits de un cromosoma sean mutados. La tabla 2.4 muestra una posible mutación en el cromosoma A'_1 del descendiente del individuo 1, para una probabilidad de mutación $Pm = 0,001$. Para que la mutación ocurra, el valor del número aleatorio de 4 decimales deberá encontrarse entre $0,0000$ y $0,0009$.

Bit	Número aleatorio	Mutación %
1	0,1095	no
2	0,9505	no
3	0,6968	no
4	0,0009	si
5	0,0019	no
Cromosoma	01100	
Cromosoma mutado	01110	

Tabla 2.4: Proceso de mutación

2.1.5.2. Cuadro resumen

En la tabla 2.5 se muestran conjuntamente los resultados de los operadores aplicados para obtener la primera generación, a partir de la población inicial propuesta.

Indv.	Cromosoma	Valor (s)	V. Salud	% del Total	Resultado de ruleta	Padres	Parejas
1	01101	13	169	14.4	0,020	1	
2	11000	24	576	49.2	0,382	2	1-2
3	01000	8	64	5.5	0,889	4	
4	10011	19	361	30.9	0,524	2	4-2

	N. aleatorio	h	Cruce	N. aleatorios	Mutación	Descendencia	Valor (s)
1	90221	4	01100	Ver	si	01110	14
2		4	11001	tabla	no	11001	25
4	49377	2	10000	2.6	no	10000	16
2		2	11011		no	11011	27

Tabla 2.5: Cuadro resumen GAs

Bit	N. aleatorio(1)	N. aleatorio (2)	N. aleatorio (3)	N. aleatorio (4)
1	0,1095	0,6871	0,9310	0,6095
2	0,9505	0,2633	0,2150	0,9057
3	0,6968	0,5785	0,4316	0,6696
4	0,0009	0,7958	0,3586	0,7834
5	0,0019	0,0609	0,3637	0,0106
Cromosoma	01100	11001	10000	11011
Cromosoma mutado	01110			

Tabla 2.6: Mutación. Números aleatorios

2.2. Fórmulas electorales

Se entiende por *fórmula electoral* el procedimiento matemático que se aplica para la conversión de votos en escaños, esto es, para el reparto de los mismos entre las distintas candidaturas. Es también conocido como modo de escrutinio o fórmula de reparto de escaños (en inglés proportional representation-PR). La fórmula electoral identifica particularmente la modalidad del sistema de decisión mayoritaria o proporcional [CS92].

2.2.1. Fórmulas mayoritarias

Dentro de ellas suelen diferenciarse básicamente las siguientes [CS92]:

2.2.1.1. Fórmula de mayoría simple o relativa

Es la fórmula mayoritaria por excelencia: ganan los candidatos que obtienen mayor número de votos, con independencia de cualquier otra circunstancia.

2.2.1.2. Fórmula de mayoría absoluta

Intentan corregir las distorsiones provocadas por las fórmulas de mayoría relativa, a través de la exigencia de la mayoría absoluta de votos emitidos (la mitad más uno). Entre ellas se distinguen:

Fórmula mayoritaria a dos vueltas. La elección se decide, en principio, por mayoría absoluta, pero si ningún candidato la alcanza, se celebra una segunda votación (segunda vuelta) en la que basta la mayoría simple. Normalmente, se restringe la participación de candidatos en la segunda vuelta a quienes en la primera hubieran obtenido el voto de cierto porcentaje de electores o a los dos candidatos con mayor número de votos.

Voto alternativo. Forma de voto ordinal que pretende reunir en una las dos operaciones de la fórmula de la segunda vuelta. El elector fija en su voto un orden de preferencias entre candidatos y cuando, una vez contabilizadas las primeras preferencias, éstos no consiguen la mayoría absoluta, se acude a eliminar sucesivamente a los candidatos con peores resultados y a contabilizar las segundas preferencias de quienes les habían votado, hasta que haya candidatos que alcancen mayoría absoluta.

2.2.2. Fórmulas distributivas o proporcionales

Las fórmulas distributivas ([Pas91] y [CG97]) responden al criterio de que la asignación de escaños debe *distribuirse* en función de la cantidad de votos obtenidos por cada candidato o partido, en lugar de atribuirlos directamente al que obtiene mayor número de sufragios.

2.2.2.1. Fórmulas de cociente electoral común

Se trata de una amplia familia de fórmulas que recurren primeramente a establecer el número de votos que da derecho a un escaño y, a partir de ahí, calcular el número de puestos que corresponderá a cada partido. Llamaremos Q a la cuota o número de votos necesarios para conseguir un escaño, la cual vendrá determinada a través de un cociente. El cociente más sencillo es aquel que resulta de dividir la suma de votos válidos emitidos V entre el correspondiente número de escaños a elegir M. Es el llamado *cociente entero* o de Hare:

$$Q_h = \frac{V}{M} \tag{2.2}$$

Existen además otras variantes que aumentan en una o más unidades el valor del denominador de esta. La tabla 2.7, muestra el conjunto de todas estas variantes.

Variantes	Cálculo de la cuota
Hare	*q=v/m*
Droop (Hagenbach- Bischof)	*q=v/(m+1)*
Imperiali	*q=v/(m+2)*
Imperiali reforzada	*q=v/(m+3)*

Tabla 2.7: Cálculo de la cuota electoral en variantes de la fórmula distributiva del resto mayor [CG97]

La adjudicación de escaños prosigue mediante la división de los votos obtenidos por cada candidatura V_a por la cuota o cociente electoral Q para conseguir el número de escaños que le corresponde E_a. Pero esta división raramente tendrá como resultado un número entero, quedando normalmente un número racional. Por lo tanto, pueden adjudicarse los escaños con arreglo a los números enteros de cada partido, pero queda pendiente la asignación de escaños restantes y la aplicación de las fracciones de cada partido o candidatura.

Para resolver este problema, dos variantes principales de fórmula distributiva nos ofrecen alternativas para solventar el problema: la *fórmula del resto mayor* y la *fórmula de la media mayor*.

1. Fórmula del resto mayor

   En esta variante distributiva, el criterio para la asignación de escaños no atribuidos en una primera distribución se basa en la magnitud del resto de los sufragios no utilizados que presenta cada candidatura.

   Una vez fijada la correspondiente cuota Hare, se atribuyen a cada lista tantos escaños como veces se contiene la cuota o cociente electoral en el total de votos que dicha lista ha recibido. Si, efectuada esta operación, quedan todavía puestos por cubrir, será la magnitud de los votos sobrantes lo que decida la distribución de estos escaños restantes. Para ello, estos escaños pendientes se irán asignando a las listas que presenten la fracción más elevada de la cuota: resto mayor.

   Con respecto a otras fórmulas, las fórmulas distributivas de resto mayor son en un principio las más favorables a una distribución de escaños que refleje más fielmente la distribución de votos entre candidaturas. Esto hace que se presente un beneficio a los partidos pequeños, que con poca representación en votos podrían alcanzar algún escaño. Por esta razón se suelen acompañar de algunas disposiciones como por ejemplo imponer una barrera electoral, que impide la participación en el reparto de escaños a partidos que no superen un determinado volumen de sufragios.

   La variante Hare-Andrae A esta variante se la conoce también como "voto único transferible" debido a su asociación con determinada modalidad preferencial de voto. En esta, cada elector vota por su candidato preferido, aunque puede marcar a continuación un orden de preferencias entre los demás aspirantes.

   Al igual que en el caso Hare, se establece en primer lugar la cuota necesaria para obtener un escaño. El recuento de las primeras preferencias obtenidas por cada uno de los candidatos nos dirá cuáles de entre ellos alcanzan dicha cuota y resultan inmediatamente elegidos.

   Si después de esta primera adjudicación siguen vacantes algunos escaños, los sufragios excedentes a favor de los candidatos ya elegidos se adjudican a los candidatos que constataban en sus votos como segunda preferencia del elector. Cuando el candidato elegido no disponga de sufragios excedentes, serán las segundas preferencias del candidato situado en el último lugar las

que se utilizarán ahora para favorecer a otros en mejor situación. En ambos casos se *transfieren* los votos "innecesarios" para su primer beneficiario. Serán útiles, en cambio, para que otros candidatos alcancen la cuota y puedan ser elegidos. Cuando después de esta segunda etapa permanecen todavía algunos escaños por proveer, se repite la operación tantas veces como sea necesario.

2. Fórmula de la media mayor

   Este tipo de fórmulas tiene como fin conseguir que el coste medio ("media") en votos a pagar por conseguir un escaño, sea sensiblemente el mismo para cada partido. Para ello, cada uno de los escaños en liza va siendo atribuido sucesivamente al partido o lista que representa una media más elevada de votos por escaño.

   La versión más simple de esta fórmula opera del siguiente modo. En una primera fase, se establece la cuota o cociente electoral (cuota Hare) descrita en esta sección. A continuación se procede a una primera atribución de escaños, otorgando a cada partido tantos escaños cuantas veces se contiene la cuota en su total de sufragios.

   Si tras esta primera fase, permanecen escaños sin provisión y los partidos conservan votos sobrantes, se procede a una segunda distribución. Para ello, los votos de cada partido se dividen por el número de puestos que ya ha conseguido en la primera fase más uno, con el fin de averiguar cuál sería su media de votos por escaño si se le atribuyera el puesto pendiente de provisión.

   El partido que presenta una *media mayor* se hace entonces con el escaño. Si permanece todavía algún escaño no provisto, se procede de nuevo a obtener las medidas de cada candidatura, de acuerdo con la operación descrita.

   La fórmula de la media mayor suele favorecer a los partidos o candidaturas con mayor número de votos y perjudica a los minoritarios, puesto que son aquellos los que suelen presentar mejores medias en las sucesivas etapas.

La variante Hagenbach-Bischoff Esta fórmula de asignación de escaños está basada en la media mayor y presenta una utilización de la cuota Droop, conocida también como Hagenbach-Bischof.

La asignación de escaños se produce en dos fases. Primero se obtiene la cuota electoral, dividiendo el total de votos válidos emitidos V por el número de escaños en disputa M más uno (ver expresión 2.3) A continuación,

el total de votos de cada partido es dividido por esta cuota Q_r obteniendo así el número de escaños que ya le hubieran sido asignado en la primera distribución incrementado en una unidad. Se procede nuevamente de este modo hasta completar la asignación de todos los puestos.

$$Q_r = \frac{V}{M+1} \tag{2.3}$$

2.2.2.2. Fórmulas de divisores comunes

En ellas, no hay varias fases ni reparto de restos. Toda su operación se concreta en dividir la cifra de votos que ha obtenido cada partido por una serie determinada de números, de lo cual resulta la correspondiente serie de cocientes para cada uno de los partidos; a continuación se procede a atribuir los *n* escaños en juego a los *n* cocientes más altos, esto es, un partido logra tantos escaños como cocientes mayores haya conseguido.

Este tipo de fórmulas (especialmente la ley d'Hondt y la Imperiali) tienden a beneficiar a los partidos mayores y a disminuir la ventaja relativa que los partidos pequeños pueden obtener con la fórmula del resto más elevado.

La variante d'Hondt Lleva el nombre del profesor Belga que la ideó. El proceso consiste en dividir el número de sufragios de cada candidatura por 1-2-3-4, etc., hasta el número total de escaños a cubrir. A continuación los cocientes resultantes de las divisiones anteriores se ordenan de mayor a menor, en una serie que ha de comprender tantos cocientes como escaños en disputa.

A partir de ese momento, se atribuyen los escaños a los partidos o listas que representan los mayores cocientes de la lista.

Es utilizada actualmente en varios países. En España se aplica con una barrera electoral del 3 %, es decir, no se tienen en cuenta aquellas candidaturas que no hubieran obtenido al menos el 3 % de los votos válidos emitidos en la circunscripción.

La variante Saint-Laguë Algunos sistemas electorales incorporan modificaciones en la forma de calcular la media de votos de cada partido, con el fin de atenuar la desigualdad que ofrecen las fórmulas de divisores comunes.

Esta variante recurre a sustituir los divisores 1-2-3, etc., por la serie de números impares 1-3-5, etc. De este modo, al incrementar la diferencia entre divisores, se

incrementa además la diferencia entre cocientes. Con ello los partidos mayores ven relativamente disminuida su ventaja.

La variante Saint-Laguë modificada Sustituye en el primer divisor el 1 por 1,4 (1.4-3-5-7, etc.)

La variante Imperiali Presenta un sentido opuesto a la variante Sain-Laguë con el fin de promocionar a los partidos más grandes, estableciendo la serie de divisores 2-3-4-5, etc. Al excluir el divisor 1, se hace muy difícil obtener el primer escaño a los partidos o candidaturas con menos votos y se aumenta el peso de los partidos mayores.

No se debe confundir esta variante, con la cuota Imperali vista en el apartado 2.2.2.1.

2.2.2.3. Fórmulas de cociente electoral de lista

Son aquellas que resuelven la adjudicación de los escaños en juego a partir del cálculo de sendos cocientes específicos de partido, operando con los resultados electorales obtenidos por cada uno de ellos.

La variante Hare-Niemeyer Esta variante, responde a la misma lógica de la fórmula del resto mayor con cuota Hare. Para obtener la cuota, se multiplica primero el número de votos conseguido por cada partido V_a por la magnitud o número de escaños M. A continuación, se divide la cantidad resultante de esta multiplicación por el total de votos válidos emitidos V. La cuota por partido así obtenida nos ofrece el número de escaños que se le asignan, distribuyendo primero los correspondientes a cuotas enteras y asignando los restantes a partidos con mayores fracciones de cuota.

$$Q_a = \frac{V_a \cdot M}{V} \tag{2.4}$$

Capítulo 3

Alternativas y solución tomada

Una vez introducido el proyecto y presentado el marco teórico, se procederá a abordar los objetivos. El proyecto se compone de cuatro objetivos instrumentales enfocados para poder alcanzar el objetivo principal. Estos consisten en determinar qué fórmula electoral de divisores comunes se empleará en la exploración, encontrar un software de algoritmos genéticos y disponer de un conjunto de funciones test y procedimientos de evaluación.

A continuación se recoge una recopilación de software de computación evolutiva, baterías funciones test y procedimientos de evaluación. Además se realiza y se justifica la selección tanto del software, procedimiento de evaluación y función objetivo así como de la fórmula electoral, que posteriormente se utilizarán para realizar la experimentación.

3.1. Búsqueda y selección de software de computación evolutiva

El carácter cíclico y la fácil implementación computacional de los algoritmos genéticos, hace que estos estén pensados para su ejecución en un ordenador, por lo que intentar trabajar con ellos en el papel, carece de fundamento. Es por ello que una de las primeras fases consiste en la búsqueda y selección de software que permitiese la ejecución de algoritmos genéticos, dentro del software de computación evolutiva, con el fin de alcanzar el objetivo instrumental 4 fijado.

3.1.1. Búsqueda

En primer lugar, se ha llevado a cabo una exhaustiva búsqueda en la red, de aquellas aplicaciones que emplearan técnicas evolutivas para resolver problemas

de optimización, software de computación evolutiva, poniendo especial interés en aquellas que usaban los algoritmos genéticos.

El objetivo de esta búsqueda es el de encontrar una aplicación informática que permitiese:

1. Poder ejecutarlo con la configuración del AGS.

2. Personalizar sus operadores, para así poder implementar como operador de selección la fórmula electoral que escogeremos.

3. Presentación de resultados en un archivo de texto o en su defecto en otro formato que permita fácilmente su representación, siempre y cuando no lo realice directamente.

4. Código accesible para poder programar y utilizar el nuevo operador selección.

Tras la búsqueda se ha tenido en cuenta el siguiente software:

- SGA-C[1] [SGE94]: es la implementación en lenguaje-C del algoritmo genético Simple presentado en código Pascal por Goldberg en su libro [Gol89a]. Aunque presenta características adicionales, su funcionamiento es el mismo que la versión original en Pascal.

- GA toolbox[2] [Sas07]: este toolbox ha sido implementado en C++ por Illigal[3]. Se trata de un software de algoritmos genéticos concebido tanto para problemas de optimización mono-objetivo como multi-objetivo. Además, permite elegir entre diferentes operadores y otras herramientas para la ejecución del algoritmo genético.

- LibGA100[4] [CW95]: librería de algoritmos genéticos desarrollada en C por Arthur L. Corcoran para la resolución de problemas de optimización combinacional. LibGA incluye un variado set de operadores genéticos de selección, cruce y mutación, para su selección a la hora de ejecutarlo, así como otras herramientas avanzadas.

[1]Fuente: http://www.illigal.uiuc.edu/pub/src/simpleGA/C/sga-c.tar.Z

[2]Fuente: ftp://ftp-illigal.ge.uiuc.edu/pub/src/GA/GAtoolbox.tgz

[3]Illigal (Illinois Genetic Algorithms Laboratory) es un laboratorio ubicado en la universidad de Illinois y dirigido por David E. Goldberg. Se dedica a realizar estudios sobre selección de algoritmos naturales de búsqueda, algoritmos genéticos y evolutivos, así como también enfoques prácticos para la resolución de problemas en ordenador.

[4]Fuente: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/libga/

- SES 1.0[5]: SES (Simple Evolution Strategy) es un programa de optimización de parámetros implementado por Joachim Sprave, basado en Estrategias Evolutivas (Evolution Strategies (ES))[Qua98].

- VectorGA[6]: se trata de la implementación vectorizada en código de Matlab de un algoritmo genético. Su autor, Keki Burjorjee, comenta en su blog [7] que la implementación vectorizada (la vetorización consiste en sustituir los bucles de la programación por arrays), consigue aumentar el rendimiento de este programa al ejecutarlo en Matlab. Existe un link desde la página oficial de Illigal al mencionado blog.

La tabla 3.1 muestra una comparativa de las principales características del software de computación evolutiva anteriormente descrito. En esta se ha reflejado aspectos tan importantes como el sistema operativo para el que han sido implementadas, de qué tipo de aplicación se trata o si permite operar con interface.

Como se puede comprobar, cumplen los 4 requisitos mencionados en este apartado ya que:

- Permiten la ejecución del AGS, ya sea directamente, configurándolas en su inputfile o cambiando el código.

- Son personalizables. Al tratarse de código abierto, unas permiten simplemente cambiar el código y otras añadir operadores nuevos y nuevas herramientas, según de cual se trate.

- A excepción de SES, que muestra los resultados por interface junto con representaciones gráficas, el resto permite salvar estos en un outputfile, facilitando el manejo.

3.1.2. Selección

Una vez terminada la búsqueda, se llevó a cabo la selección de las alternativas descritas en el anterior apartado. Para ello se ha recurrido a la decisión multicriterio por medio de los siguientes métodos basados en la ponderación [BRP97]: método de las sumas ponderadas, método TOPSIS y método ELECTRE. En el apéndice A puede consultarse el desarrollo completo de estos métodos aplicados a esta nuestra particular selección.

[5]Fuente: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/ses
[6]Fuente: http://evoadaptation.wordpress.com/2007/01/25/january-23-2007-untitled/
[7]Fuente: http://blog.hackingevolution.net/2007/01/25/vector-ga/

Software	S.O.	Funcionamiento	Tipo	AGS	Configurable
SGA-C	Linux /Windows	Ejecutable	Ejecutable C	Solo programado para AGS	Modificando el código
GAtoolbox	Linux /Windows	Ejecutable	Toolbox en C++	Si con pequeñas modificaciones de código	Si, en el inputfile
LibGA	Linux /Windows	Ejecutable	Librería	Si con algunas modificaciones de código	Si, en el config.c
SES	Linux /Windows	Ejecutable	Ejecutable C	Cambiando el cógido	Modificando el código
VectorGA		Matlab	M-file de Matlab	Si, sobre el código	Modificando el código
	Personalizable	Interface	Entrada de parámetros	Salida de resultados	Representación
SGA-C	Si	No	Inputfile y terminal	Outputfile o terminal	GNUplot
GA toolbox	Si	No	Inputfile	Outputfile + terminal	GNUplot
LibGA	Si	No	En código	Outputfile + terminal	GNUplot
SES	Si	Si	Interface	Outputfile + Gnuplot	GNUplot
VectorGA	Si	No	En código	MATLAB Command Window	MATLAB plot

Tabla 3.1: Comparativa de las aplicaciones informáticas

3.1.2.1. Criterios para la selección

A continuación se describen los criterios (solamente cualitativos) que se han considerado necesarios para la selección de la mejor alternativa, es decir del paquete informático que mejor se adapta a las necesidades de este proyecto.

- Configurable (C_1): indica la posibilidad de la aplicación para seleccionar diferentes operadores y herramientas.
- Adaptabilidad a AGS (C_2): indica la facilidad de seleccionar o programar los distintos operadores para ejecutar el AGS en la aplicación.
- Facilidad de uso
  - Interface y definición de parámetros (C_3): existencia de interface y modo en el que se introducen los parámetros y selección de opciones.
  - Salida y representación (C_4): modo en el que la aplicación muestra los resultados y su representación gráfica.
- Fuente (C_5): procedencia de la aplicación informática. Se ha incluido este criterio debido a que según la procedencia se brindan servicios de soporte, foros de preguntas... Además, cuanto más importante o prestigiosa sea la procedencia de la aplicación, más extendido estará su uso lo que implica una mayor facilidad de encontrar información de esta en internet.

3.1.2.2. Resultados

En el apéndice A se encuentra el desarrollo de los 3 métodos multicriterio: sumas ponderas, TOPSIS y ELECTRE para determinar qué software de los indicados en este capítulo se va a utilizar para llevar a cabo este proyecto. La tabla 3.2 muestra un cuadro que refleja los resultados obtenidos en cada método.

Analizando estos resultados, según el método de las sumas ponderas y TOPSIS, se escogería la alternativa 2 (GA toolbox). Para el método ELECTRE y con un límite de concordancia $S_c = 0,65$ y de discordancia $S_c = 0,29$ la alternativa preferida sería la alternativa 2. Por lo tanto el software de computación evolutiva a utilizar, bajo el punto de vista de los tres criterios, se trata sin duda del GA toolbox, que como se ha comentado, es un software de algoritmos genéticos.

Con la elección del software hemos cumplido en parte el objetivo instrumental 4. En la definición de este objetivo se indica que el software de computación evolutiva debe permitir la ejecución del algoritmo genético simple. Como se ha

Métodos \ Alternativas		A1	A2	A3	A4	A5
Sumas ponderadas						
	V_i	0,25	0,3	0,19	0,08	0,18
TOPSIS						
	C_i	0,67	0,83	0,51	0,09	0,45
ELECTRE						
	$S_c = 0,65$ $S_d = 0,29$	0	1	0	0	0

Tabla 3.2: Matriz de Decisión y pesos

reflejado en la tabla 3.1, para disponer del AGS en GA toolbox son necesarias modificaciones de código. Estas se muestran en la sección 4.2.

3.2. Funciones test y procedimientos de evaluación

Dentro de la bibliografía así como en la red, existen diversas baterías de funciones test y procedimientos de evaluación (consultar apéndice D), que van desde algunas funciones test hasta conjuntos de varias funciones con procedimiento de evaluación asociado (denominados test suites). De entre ellos, con el fin de abordar los objetivos instrumentales 2 y 3 (ver sección 1.2), a continuación se muestran las que han considerado más relevantes para este proyecto y acordes con lo descrito en el apartado D.1.0.5 del mencionado apéndice, además de seleccionar el problema de optimización o función objetivo a utilizar.

3.2.1. Selección de la batería de funciones test y el procedimiento de evaluación

- Estudio de De Jong [Gol89a]: consta de 5 funciones de minimización diseñadas por De Jong.
- Genetic Algorithm Toolbox-Test Functions [Poh94]: se trata de las 13 funciones que vienen implementadas en este software.
- GEATbx[8]: relación de 16 funciones test implementadas para su

[8]http://www.geatbx.com

uso en el software *Genetic and Evolutionary Algorithm Toolbox for Matlab (GEATbx)*. Estas funciones están representadas en `http://www.geatbx.com/docu/fcnindex-01.html`.

- Problem Definitions and Evaluation Criteria for the CEC 2005 [SHL+05] (en lo sucesivo PDEC-05): a diferencia de las anteriores, PDEC-05 no solo dispone de un conjunto de funciones test, sino que se acompaña conjuntamente de un procedimiento de evaluación (bastante completo) por lo que constituye una test suite como tal. Fue creada para el CEC 2005 (IEEE Conference on E-Commerce Technology) con el propósito de chequear una test suite estándar proponiendo a investigadores (probablemente, la mayoría de ellos asistirían a la conferencia) que trabajaran con ella resolviendo sus funciones de optimización, para mostrar los resultados en la mencionada conferencia. El documento que detalla el test se puede descargar de internet y además se ha adjuntado en el apéndice D.

En relación a satisfacer estos objetivos instrumentales, el famoso estudio de De Jong, las funciones de GEATbx y Genetic Algorithm Toolbox (las tres primeras posibilidades) solamente nos aportan funciones test, pero no ofrecen ningún procedimiento de evaluación. Sin embargo la última posibilidad, PDEC-05, no solamente dispone de una colección de funciones, sino que también describe un procedimiento de evaluación para analizar los resultados que devuelva el algoritmo genético ante distintos escenarios (distintas funciones, parámetros, número de evaluaciones...), es decir, una test suite como tal. Este procedimiento analiza todas las condiciones descritas en el apartado D.1.0.6 del apéndice D (consultar sección 4.3).

Además, de la página web[9] del PDEC-05, se puede descargar un software en el que se encuentran implementadas todas las funciones en C, lo cual lo hace compatible con el software seleccionado en la sección 3.1 y que posiblemente se pueda "acoplar" con relativa facilidad. Por otro lado, el conjunto de las 25 funciones test que ofrece, está compuesto en su mayoría por funciones que se incluyen en las tres primeras posibilidades, lo que hace que el conjunto englobe todo y asegure poder satisfacer los aspectos mencionados en el apartado D.1.0.5. Por último, aplica una serie de desplazamientos y rotaciones del óptimo global, inserción de ruido, óptimos en la frontera, funciones compuestas y funciones híbridas... que intentan que el algoritmo muestre sus características ante todos los escenarios posibles y no empeore con respecto al test al aplicarlo a otros problemas (apartado D.2).

[9] http://web.mysites.ntu.edu.sg/epnsugan/PublicSite/Shared Documents/Forms/AllItems.aspx

Cabe destacar que acorde con las sucesivas ediciones del CEC, desde el 2005 se proponen nuevas especificaciones con distintos objetivos. Esto otorga consolidación al procedimiento así como su aceptación y difusión. Por lo tanto, bajo estos aspectos, es el PDEC-05 la test suite escogida para el estudio de nuestro algoritmo genético. A continuación se enumeran de nuevo los motivos:

1. Oferta tanto conjunto de funciones test como procedimiento de evaluación.
2. Amplio conjunto de 25 funciones test.
3. Funciones implementadas y con gran popularidad. Procedimiento de importante difusión.
4. Contempla y ofrece solución a los problemas en funciones test con restricciones.

Análisis del procedimiento de evaluación Adicionalmente se ha chequeado si el procedimiento de evaluación propuesto en el PDEC alcanza las siguientes condiciones:

1. Convergencia hacia un óptimo local/global, como por ejemplo número de evaluaciones para alcanzar un óptimo.
2. Rendimiento evaluación tras evaluación (mejora en curso).
3. Diferencia en la finalización con el óptimo global (conocido).

Estas condiciones fueron mostradas en el apartado D.1.0.6 del apéndice D. Los criterios de evaluación propuestos en el procedimiento del PDEC-05 son:

1. Salvar el valor de error de la función $(f(x) - f(x^*)$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (Function Evaluations) y en la finalización para cada evolución (dada por Ter_Err o Max_FES).
2. Salvar el número máximo de evoluciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión.
3. Ratio de éxito y rendimiento de éxito para cada problema.
4. Gráficas de convergencia.
5. Algorithm Complexity

La condición 1 se ve cubierta por el criterio 4, las gráficas de convergencia. El rendimiento evaluación tras evaluación (condición 2) se cumple con los criterios 2, 3 y en parte por el 1. Por último la condición 1 se ve cubierta por el criterio 1 (su valor final).

Además se añade un criterio extra (criterio 5, Algorithm Complexity) que refleja una característica más sobre el comportamiento del algoritmo.

3.2.2. Selección y descripción de la función objetivo escogida

En este apartado tiene como misión completar el objetivo instrumental 2 que como ya se comentó en la sección 1.2, no solo consiste en determinar la batería de problemas sino que además hay que seleccionar el problema de optimización a utilizar. De entre las 25 funciones test que ofrece el PDEC-05, la test suite escogida, ha sido la función 1, función Esfera desplazada, la que nos ha parecido más idónea para la comparación, ya que es de sobra una función conocida, que aparece en muchos textos y que es la primera función del conjunto de funciones de De Jong. A continuación se describe la función.

f_1: Función Esfera desplazada (De Jong 1 desplazada)

$$F_1(\mathbf{x}) = \sum_{i=1}^{D} {z_i}^2 + f\_bias_1; \tag{3.1}$$

donde: $\mathbf{z} = \mathbf{x} - \mathbf{o}$, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$
$D : dimensiones$ $\mathbf{o} = [o_1, o_2, \ldots, o_D] : el\ óptimo\ global\ desplazado$

Propiedades:

- Unimodal.
- Desplazada.
- Separable.
- Escalable.
- $\mathbf{x} \in [-100, 100]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_1(\mathbf{x}^*) = f\_bias_1 = -450$.

Con la elección del PDEC-05 y la selección de la función objetivo quedan cubiertos los objetivos instrumentales 2 y 3.

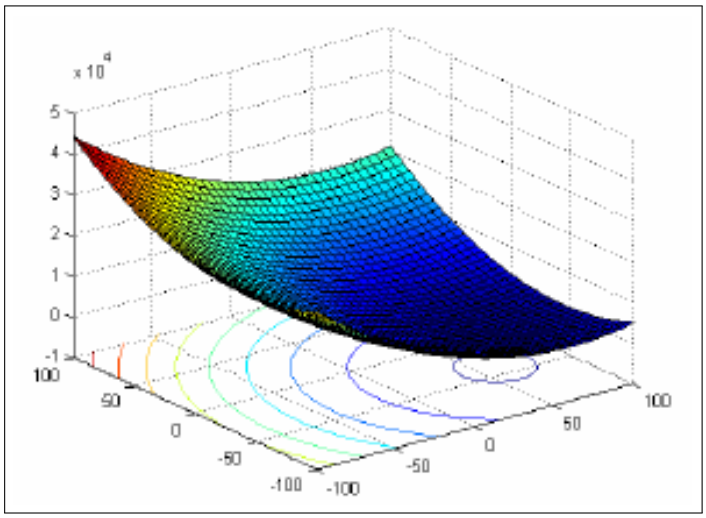

Figura 3.1: Representación invertida 3D de la función F1 en 2D [SHL+05]

3.3. Alternativas de fórmulas de divisores comunes

Como se vio en la introducción (capítulo 1), de entre las fórmulas electorales presentadas en la sección 2.2, son las mayoritarias y las de cociente electoral común las que de algún modo han visto reflejado su fundamento en diversas técnicas heurísticas usadas en métodos de optimización. El primer apartado de esta sección proporciona un ejemplo de ello.

Sin embargo, para las fórmulas de divisores comunes, las más empleadas en los sistemas de gobierno actuales, se desconoce si han servido como base en el operador selección de los algoritmos genéticos. A partir de ello, lo que se quiere obtener con el objetivo instrumental 1 es determinar cuál de estas fórmulas será la empleada como operador selección en este proyecto. Este objetivo se lleva a cabo en esta sección.

3.3.1. Empleo de fórmulas electorales en el valor esperado

Como justificación de este empleo, podemos mencionar que en los algoritmos genéticos, el método de selección del *valor esperado*, presentado como alternativa de selección en el apartado 2.1.3, muestra mucha similitud con las *fórmulas de cociente electoral común*, que se vieron en el apartado 2.2.2.1, utilizadas en el reparto de escaños electoral. Así pues, en el valor esperado, se calcula para cada individuo su valor de salud normalizado y se multiplica por el número de descendientes esperado, lo que viene a ser similar a calcular el *cociente entero*

o de Hare. Luego, al igual que en las fórmulas de cociente electoral común, se reparten el nº de hijos con arreglo a los números enteros de estos cocientes (por ejemplo, si un individuo tiene un nº de $2,47$, por lo menos será dos veces elegido para ser padre de la siguiente generación). Pero la cosa no queda solo ahí ya que para repartir el resto, una de las variantes que ofrece el valor esperado, es el reparto determinista, que no es otra cosa que asignar el resto de hijos, según la fracción más elevada del cociente, lo que coincide con la *fórmula de resto mayor* que acompaña a las fórmulas de cociente electoral común, para el reparto de restos. La siguiente figura 3.2 contiene un esquema que describe estas similitudes.

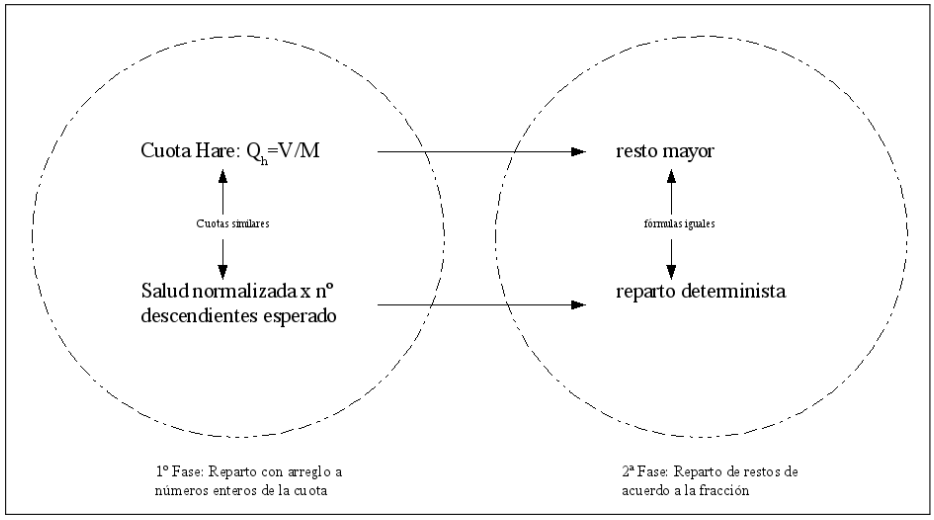

Figura 3.2: Similitudes valor esperado-cociente electoral común

3.3.2. Selección de la fórmula de divisores comunes

De entre las fórmulas de divisores comunes, en el apartado 2.2.2.2 se refleja que la variante d'Hondt, Saint-Lagüe, Saint-Lagüe modificada y la Imperiali son las más conocidas de este grupo. Analizando el comportamiento de cada una de ellas en el supuesto del apéndice B tenemos que las variantes de Sainte-Laguë disminuyen la ventaja de los partidos que más votos obtienen y favorecen a partidos con número de votos intermedios. A su contra, las variantes d'Hondt e Imperiali son las que muestran favoritismo hacia los partidos más fuertes (sobre todo la Imperiali) o con más votos, dejando sin representación a los partidos más débiles o con menos votos.

El método d'Hondt se caracteriza por ser una de las fórmulas más populares en Europa y, en particular, la que se emplea en el sistema electoral español. Además:

- Es un método de operatividad fácil y de efectos previsibles.
- En un solo procedimiento basta para poder distribuir todos los escaños (gran ventaja frente a los otros métodos).

Para la selección se han tenido más en cuenta las fórmulas conservadoras, ya que entendemos que pueden ser más eficaces en algoritmos genéticos. Dado que la Imperiali podría resultar demasiado conservadora y con las características descritas, se ha elegido al método d'Hondt, como el método que vamos a probar como operador selección en algoritmos genéticos, alcanzando así el objetivo instrumental 1 planteado en la introducción.

Capítulo 4

Desarrollo

En la introducción se ha planteado el objetivo principal que consiste básicamente en comparar en el AGS el operador ruleta frente al nuevo operador selección basado en la fórmula electoral de divisores comunes escogida. Uno de los aspectos más importantes para poderlo llevar a cabo, es disponer en el software de computación evolutiva elegido, GA toolbox, del nuevo operador. A parte, tal como se comentó en el capítulo 3, GA toolbox necesita algunas modificaciones para ejecutar el AGS (algoritmo genético simple). Adicionalmente son requeridas una serie de operaciones y programaciones de cara a la experimentación. Estas operaciones son la implementación de la test suite *PDEC-05* -función y procedimiento- además de la creación de un nuevo programa para la recopilación de los datos, *Statistics*.

De cara a conocer su funcionamiento, opciones y ficheros que lo componen se realiza en primer lugar una descripción del software GA toolbox. Esta descripción será útil para abordar a continuación el desarrollo de GA toolbox, que comprende soluciones tomadas para la creación de nuevos operadores, la implementación del operador selección según la fórmula electoral escogida (ley d'Hondt), codificación, implementación de las funciones y el código modificado. El nuevo software resultante recibirá el nombre de MGA toolbox (Modified GA toolbox). Como cierre, se muestra como se ha implementado el programa, Statistics, diseñado acorde con GA toolbox y los criterios del PDEC-05.

Lo que se pretende con en este capítulo es tener disponible el software, tanto de algoritmos genéticos como de recopilación de datos, para la siguiente fase, la experimentación.

4.1. GA toolbox: Single and Multiobjective Genetic Algorithm Toolbox

En el capítulo anterior, cumpliendo con el objetivo instrumental 4 se ha seleccionado el software de computación evolutiva a emplear: GA toolbox.

Se trata un software de optimización por algoritmos genéticos para problemas de optimización mono- y multi-objetivo que se puede obtener gratuitamente en: `ftp://ftp-illigal.ge.uiuc.edu/pub/src/GA/GAtoolbox.tgz`. En el apéndice C de esta memoria se incluye el reporte técnico [Sas07] desarrollado por los programadores del software GA toolbox.

Este toolbox dispone de diferentes operadores de selección, cruce, mutación, niching y gestión de restricciones. Además, al ser de código accesible, permite ser personalizado de un modo fácil, para la modificación/adicción de operadores definidos por el usuario.

El toolbox ha sido implementado en C++ por Kumara Sastry del Laboratorio de Algoritmos Genéticos de la Universidad de Illinois (IlliGAL [1]). Dicho laboratorio está dirigido por David E. Goldberg lo que sin duda garantiza que el software utiliza las técnicas originales de los algoritmos genéticos así como que dispone de los últimos avances.

4.1.1. Características y opciones

GA toolbox ofrece un amplio abanico de operadores así como distintas opciones, tales como representación, salida a fichero, aproximación... A continuación se presenta una breve descripción de estas opciones, aconsejándose acudir a la documentación en caso de que se desee conocer más sobre el tema en cuestión.

Tipo de optimización GA toolbox permite resolver problemas de optimización tanto mono- como multi-objetivo. Para la resolución de problemas multiobjetivo recurre al *non-dominated sorting GA-II* (NSGA-II) [KPAM02].

Codificación de variables Las variables de decisión con las que trabaja el software son codificadas como *reales* o *enteros* haciéndose necesario también la definición de los valores limites que tomarían. En la bibliografía [DB99], se explica

[1] http://www.illigal.uiuc.edu/web/

cuales son los motivos de la codificación "real". Entre otros destacan la versatilidad y la carencia de cadenas de longitud fija que condicionan la precisión.

Operador Selección

- Torneo con reemplazamiento [Gol89a], [SG01]. En este método un número específico de individuos, s, es seleccionado de la población actual (formada con n individuos). Estos n individuos se enfrentan entre ellos (en un torneo). El mejor consigue ser padre de la siguiente generación. En este caso los candidatos seleccionados en un torneo, son candidatos en otros torneos (reemplazamiento). Este proceso se repite hasta obtener los n individuos que forman el conjunto de padres.

- Torneo sin reemplazamiento [Gol89a], [SG01]. Igual que el anterior, solamente que los candidatos que han sido seleccionados en un torneo, no pueden ser seleccionados para participar en otros torneos (sin reemplazamiento). En caso de que todos los individuos hayan participado en torneo y no se haya alcanzado los n individuos del conjunto de padres, se marcan todos como "no participados" y se continúan los torneos.

- Truncado [MSV93]. Los candidatos son ordenados según su valor de salud. Se elige una proporción p $(1/2, 1/3...)$ de individuos que harán el conjunto de padres. En el operador cruce se reproducen con la porción $1/p$.

- Ruleta [Gol89a]. Explicado en el apartado 2.1.5.

- Stochastic universal selection (SUS) [Bak85], [Bak87], [GB89], [Gol89a]. Los individuos son localizados en segmentos contiguos a lo largo de una línea, de modo que cada segmento (perteneciente a cada individuo) es representado con un tamaño igual al valor de salud de su individuo, como en la ruleta, siendo la suma de todos los segmentos igual a 1. Aquí, un número de punteros equiespaciados, igual al número de individuos a elegir, son emplazados sobre una línea. Considerando n el número de individuos del conjunto de padres de la siguiente generación (y por lo tanto a elegir), entonces la distancia entre punteros es 1/n. Se determina la posición del primer puntero por medio de un número aleatorio en el rango $[0, 1/n]$. A partir de ahí, se colocan los n diferentes punteros equiespaciados por $1/n$. Los individuos elegidos resultarán aquellos cuyo segmento coincida con un puntero.

Operador Cruce

- Cruce simple (un punto) [Gol89a], [SGK99]. Se selecciona aleatoriamente un punto de cruce y se intercambian los genes[2] que caen hacia la izquierda de este para crear dos nuevos individuos.

- Cruce múltiple de dos puntos [Gol89a], [SGK99]. En este caso, a diferencia del anterior, una pareja de individuos intercambian genes que se encuentran entre esas dos posiciones seleccionadas aleatoriamente.

- Cruce uniforme [Gol89a], [SGK99]. Consiste en formar el cromosoma del nuevo individuo cogiendo cada gen del padre o de la madre de una manera aleatoria.

- Simulated binary crossover (SBX) [DA95], [DK95] Se crean nuevos individuos basados en una distribución probabilística similar al cruce simple binario explicado en el apartado 2.1.5.

Operador Mutación

- Mutación selectiva: aleatoriamente se selecciona uno de los genes y si no está asignado como inmutable (lo que se denomina "congelado"), se remplaza por un valor escogido aleatoriamente dentro del rango de la variable que corresponde correspondiente al rango.

- Mutación genética: para cada gen se añade un valor procedente de una distribución normal con media cero y una desviación estándar definida por el usuario, siempre y cuando el gen no esté "congelado"

- Mutación polinomial[DA95], [DK95], [Deb01]: Este operador es idéntico al Simulated binary crossover (SBX) solamente que aplicado al operador mutación.

Operadores de Nicho (Niching) Estos operadores se basan en la creación de nichos o agrupamientos de individuos. Están encaminados a mantener la diversidad genética de la población por medio de estos nichos, de forma que cromosomas similares sustituyan sólo a cromosomas similares, esto es, dentro de un mismo nicho.

[2]Para el caso de GA toolbox, en gen representa la secuencia correspondiente a una variable del problema

- Función de compartición (fitness sharing) [GR87], [Gol89a]. La premisa en el caso del método de Sharing es que los individuos de características similares compartan la misma salud. Aunque permite dar ventaja en la población a individuos que presentan nuevas características sobre los que son redundantes permitiendo la creación de nuevos nichos, a su vez también mantiene los nichos actuales.
- Hacinamiento determinístico (deterministic crowding) [Mah92]. Todos los individuos de la población son emparejados aleatoriamente y recombinados. La descendencia resultante se obtiene por torneo con su padre más cercano (en términos de distancia de fenotipo), es decir, entre individuos de un mismo nicho. Los ganadores se copian a la nueva población para la próxima generación.
- Selección por torneo restringida (restricted tournament selection) [Har95]. Para cada nuevo individuo, se selecciona aleatoriamente W individuos y de estos se busca el más cercano (distancia de fenotipo). En caso de que este sea mejor que el nuevo individuo, se remplaza directamente.

Escalado Son procesos para escalar los valores de salud de los individuos usados con operadores de selección proporcional y función de compartición.

- Ranking lineal [Bak85], [Gol89a]: selección de individuos acorde con su valor de salud y la violación de restricciones (funciones frontera), y posterior asignación a un escalado lineal como nueva función de salud. Por lo tanto el mejor individuo presentará un valor de salud de N y el peor un valor de 1.
- Escalado Sigma (Truncado) [For85], [Gol89a]: escalado de la función de salud usando:

$$f_{scaled} = 1 + \frac{f - f_{avg}}{\sigma \cdot f_{std}}; \tag{4.1}$$

donde:

f es el valor de salud de cada individuo

f_{avg} es la media de los valores de salud de los individuos

f_{std} es la desviación estándar de los valores de salud

σ es tomado como 2.0 normalmente

Si la f_{scaled} es menor o igual de 0.0 entonces se cambia este valor automáticamente a 0.1.

- Escalado por defecto (traslación): En caso de que no se determine ningún método de escalado, se tomaría el mínimo valor de salud (entre todos los individuos de la población) y se sumaría al valor de salud del resto de los individuos. Si este fuese negativo entonces se tomaría el valor absoluto.

Gestión de Restricciones Los métodos de gestión de restricciones están enfocados a penalizar a los individuos que no cumplen o cumplen parcialmente las restricciones del problema. Usan para ello, los valores de violación de restricciones (lo que falta para cumplir) en vez de los valores de la restricción en sí. Por ejemplo si tenemos la siguiente restricción:

$$10x_1 + x_2 \leq 20 \tag{4.2}$$

si $x_1 = 2$, y $x_2 = 2$, el valor de la violación de la restricción sería 2. En cambio, si $x_1 = 1$, y $x_2 = 7$, el valor de la violación de la restricción es 0.

En la gestión de restricciones, dentro del espacio de búsqueda se definen dos áreas: zona factible (se cumplen todas las restricciones) y zona no factible (no se cumple con alguna/todas las restricciones). Es importante que se pueda explorar en las regiones factibles y no factibles, debido a que algunas regiones no factibles pueden proporcionar mayor información acerca de la solución óptima que otras soluciones factibles.

GA toolbox ofrece tres métodos de gestión de restricciones: dos por penalización y uno por torneo. Para los métodos por penalización se utiliza la función $eval(x) = f(x) + p(x)$, la cual añade un término de penalización ($p(x)$) a la función objetivo por cualquier violación a las restricciones.

- Penalización lineal: dependiendo si la función es de minimización o maximización, se suma o se resta al valor de la función objetivo de un individuo una suma ponderada de las violaciones de restricciones de este individuo.
- Penalización cuadrática: similar al penalización lineal, excepto que el valor de penalización es computado como una suma ponderada del cuadrado de los valores de la violación de restricciones de una solución candidata.

El operador de selección por torneo (método de gestión de restricciones por torneo) puede ser adaptado como proceso para gestionar la violación de restricciones. A diferencia que en el operador selección aquí, el enfrentamiento entre dos individuos no solamente se tiene en cuenta el valor de la salud, sino también su factibilidad. Debido a su procedencia, solamente se puede usar cuando se selecciona los operadores de selección por torneo, truncado y SUS.

- Torneo [Deb01]: este operador trata de gestionar la violación de restricciones, basándose en el operador selección por torneo. Utiliza las siguientes reglas:

  1. Cuando dos individuos factibles son comparados, se elige el mejor de los dos
  2. Cuando un individuo factible es comparado con uno no factible, se elige el factible
  3. Cuando dos individuos no factibles son comparados, se elige el que muestre un menor valor de violación de restricciones.

Búsqueda Local El método de búsqueda local implementado en el GA toolbox es el algoritmo de *Nelder-Mead* [NM65], [PFTV89]. La búsqueda local es un método aplicado a un individuo con probabilidad p_{ls}. La solución obtenida a través de la búsqueda local se incorpora usando la estrategia de Baldwinian con probabilidad p_b, donde la solución obtenida de la búsqueda local reemplaza al individuo, pero el valor de salud del individuo no es modificado, y la estrategia de Lamarckian con probabilidad $1 - p_b$, donde el valor de salud y el individuo son reemplazados con la solución resultante de la búsqueda local y su valor de salud.

Elitismo En el GA toolbox, tanto la vieja como la nueva población es escogida de acuerdo con su valor de salud y la violación de restricciones (funciones frontera). Una parte de los individuos, p_e (definido por el usuario) son conservados y el resto $(1 - p_e)$ son remplazados por los mejores individuos de la nueva población. El tamaño de población es n.

Estadísticas En cada generación, se recogen datos de la población para elaborar estadísticas. No solamente estos datos son añadidos durante el proceso de búsqueda, sino que también se añaden diversos operadores y funcionalidades. Por lo tanto, las estadísticas son utilizadas también para determinar si la búsqueda debe terminar o no. Algunas estadísticas son la media, valor mínimo y valor máximo de salud y los valores de la función objetivo. Otras estadísticas incluyen varianza en la función de salud y objetivo en la población. Para el caso Multiobjetivo, los valores son tomados independientemente para cada objetivo.

Criterio de parada El algoritmo parará cuando se cumpla el criterio escogido (de los numerados a continuación)

- Número de evaluaciones limitado

- Varianza de los valores de salud
- El mejor valor de salud
- Media de los valores de salud
- Valor objetivo medio
- Cambio en el mejor valor salud
- Cambio en la media de los valores de salud
- Cambio en la varianza de los valores de salud
- Cambio en el mejor objetivo
- Cambio en el objetivo medio
- Número de frentes
- Número de individuos en el primer frente
- Cambio en el número de frentes

Si se elige al menos un criterio de parada es necesario determinar el número de ventana generacional. Esto es utilizado para los criterios que se basan en un cambio, de modo que si no se produce cambio en el número de generaciones marcado (ventana generacional) el algoritmo para.

4.1.2. Funcionamiento

En esta sección se muestran los aspectos necesarios para elegir las opciones que ofrece el programa GA toolbox y ejecutarlo en un ordenador. Para el proceso de compilación, por favor consulte la documentación [Sas07].

El programa lee todos los parámetros de un fichero. Por lo tanto, para poder ejecutar el programa se debe crear un fichero, como el que se muestra de ejemplo en este apartado, ubicado en el mismo directorio que el ejecutable o indicando donde se encuentra. El nombre del ejecutable es `GAtbx` (una vez compilado). El comando de ejecución por lo tanto quedaría:

```
GAtbx <input file name>
```

Si lo ejecutásemos para el fichero de entrada de este apartado tendríamos:

```
GAtbx input_sga_maxSpec
```

Para poder insertar nuestra propia función objetivo, debemos acudir al fichero `userDefinables.cpp`. Realmente este es el único fichero que hay que cambiar para insertar la función. En este fichero tenemos que buscar la función cabecera:

```
void globalEvaluate(double *x, double *objArray, double *constraintViolation,
                        double *penalty, int *noOfViolations)
```

donde:

El array x contiene las variables de una solución

El valor(es) de la función objetivo es(son) devueltos en array `objArray`

El valor(es) de la violación de restricciones se devuelve(n) en el array `constraintViolation`

El valor(es) de penalización que se añade(n) a la función de salud se devuelve(n) en el array `penalty` (solo para el caso de que se haya escogido la opción de penalización o penalización cuadrática)

Número de restricciones no cumplidas en `noOfViolatons`

Fichero de entrada A continuación se muestra un ejemplo de fichero de entrada comentado, necesario para ejecutar GA toolbox, que como se ha mencionado anteriormente, sirve para definir los parámetros y opciones del software por el usuario. El nombre de este fichero es intput_sga_maxSpec, documento simple de texto. GA toolbox salta las líneas que empiezan por # (almohadilla), que es donde se han ubicado los comentarios. Los paramentos son leídos en el orden que aparecen en este fichero, por lo que no se debe cambiar, para así obtener los resultados esperados.

```
# Input file for the GA Toolbox
# Author: Kumara Sastry
# Date: April, 2006
#

#
# GA type: SGA or NSGA
#
SGA

#
# Number of decision variables
#
2

#
# For each decision variable, enter:
#  decision variable type, Lower bound, Upper bound
# Decision variable type can be double or int
#
double 0.0 6.0
```

```
double 0.0 6.0

#
# Objectives:
# Number of objectives
# For each objective enter the optimization type: Max or Min
#
1
Min

#
# Constraints:
# Number of constraints
# For each constraint enter a penalty weight
#
2
1.0
1.0

#
# General parameters: If these parameters are not entered default
#                     values will be chosen. However you must enter
#                     "default" in the place of the parameter.
# [population size]
# [maximum generations]
# [replace proportion]
#
100
100
0.9

#
# Niching (for maintaining multiple solutions)
# To use default setting type "default"
#  Usage: Niching type, [parameter(s)...]
#  Valid Niching types and optional parameters are:
# NoNiching
# Sharing [niching radius] [scaling factor]
# RTS [Window size]
# DeterministicCrowding
#
#  When using NSGA, it must be NoNiching (OFF).
#
NoNiching

#
# Selection
#  Usage: Selection type, [parameter(s)...]
#  To use the default setting type "default"
#
#  Valid selection types and optional parameters are:
# RouletteWheel
# SUS
# TournamentWOR [tournament size]
# TournamentWR [tournament size]
# Truncation [# copies]
#
#  When using NSGA, it can be neither SUS nor RouletteWheel.
#
TournamentWOR 2

#
```

```
# Crossover
#  Crossover probability
#  To use the default setting type "default"
#
#  Usage: Crossover type, [parameter(s)...]
#  To use the default crossover method type "default"
#  Valid crossover types and optional parameters are
# OnePoint
# TwoPoint
# Uniform [genewise swap probability]
# SBX [genewise swap probability][order of the polynomial]
#
0.9
SBX 0.5 10

#
# Mutation
#  Mutation probability
#  To use the default setting type "default"
#
#  Usage: Mutation type, [parameter(s)...]
#  Valid mutation types and the optional parameters are:
# Selective
# Polynomial [order of the polynomial]
# Genewise [sigma for gene #1][sigma for gene #2]...[sigma for gene #ell]
#
0.1
Polynomial 20

#
# Scaling method
#  To use the default setting type "default"
#
#  Usage: Scaling method, [parameter(s)...]
#  Valid scaling methods and optional parameters are:
# NoScaling
# Ranking
# SigmaScaling [scaling parameter]
#
NoScaling

#
# Constraint-handling method
# To use the default setting type "default"
#
# Usage: Constraint handling method, [parameters(s)...]
# Valid constraint handling methods and optional parameters are
# NoConstraints
# Tournament
# Penalty [Linear|Quadratic]
#
Tournament

#
# Local search method
# To use the default setting type "default"
#
# Usage: localSearchMethod, [maxLocalTolerance], [maxLocalEvaluations],
# [initialLocalPenaltyParameter], [localUpdateParameter],
# [lamarckianProbability], [localSearchProbability]
#
# Valid local search methods are: NoLocalSearch and SimplexSearch
```

```
#
# For example, SimplexSearch 0.001000 20 0.500000 2.000000 0.000000 0.000000
NoLocalSearch

#
# Stopping criteria
# To use the default setting type "default"
#
# Number of stopping criterias
#
# If the number is greater than zero
#    Number of generation window
#    Stopping criterion, Criterion parameter
#
# Valid stopping criterias and the associated parameters are
# NoOfEvaluations, Maximum number of function evaluations
# FitnessVariance, Minimum fitness variance
# AverageFitness, Maximum value
# AverageObjective, Max/Min value
# ChangeInBestFitness, Minimum change
# ChangeInAvgFitness, Minimum change
# ChangeInFitnessVar, Minimum change
# ChangeInBestObjective, Minimum change
# ChangeInAvgObjective, Minimum change
# NoOfFronts (NSGA only), Minimum number
# NoOfGuysInFirstFront (NSGA only), Minimum number
# ChangeInNoOfFronts (NSGA only), Minimum change
# BestFitness (SGA with NoNiching only), Maximum value
#
0

#
# Load the initial population from a file or not
# To use the default setting type "default"
#
# Usage: Load population (0|1)
#
# For example, if you want random initialization type 0
# On the other and if you want to load the initial population from a
# file, type
#  1 <population file name> [0|1]
#
# Valid options for "Load population" are 0/1
# If you type "1" you must specify the name of the file to load the
# population from. The second optional parameter which indicates
# whether to evaluate the individuals of the loaded population or not.
0

# Save the evaluated individuals to a file
#
# To use default setting type "default".
#
# Here by default all evaluated individuals are stored and you will be
# asked for a file name later when you run the executable.
#
# Usage: Save population (0|1)
# For example, if you don't want to save the evaluated solutions type 0
# On the other and if you want to save the evaluated solutions
#  1 <save file name>
#
# Note that the evaluated solutions will be appended to the file.
#
```

```
# Valid options for "Save population" are 0/1
# If you type "1" you must specify the name of the file to save the
# population to.
1 evaluatedSolutions.txt
#END
```

4.1.3. Ficheros: Código C++

A continuación, se recoge una descripción de cada uno de los ficheros del programa GA toolbox. Cada fichero `.cpp` tiene su correspondiente fichero de cabezera `.hpp` que contienen las definiciones de las clases, exceptuando `nsgapopulation.cpp` y `userDefinables.cpp`

`chromosome.cpp` Contiene la implementación de la clase `chromosome`, la cual define al cromosoma como un array de genes. Además contiene la implementación del operador mutación.

`individual.cpp` Contiene al implementación de la clase `individual`. Un individuo es una solución dada para una determinada búsqueda y problema de optimización, el cual se compone, además del cromosoma, de más información, como por ejemplo el valor de salud, de la función objetivo...

`population.cpp` Contiene la implementación de la clase `population`, formada por un array de individuos/cromosomas. Además aquí también se han implementado "objective to fitness mappings" y "statistics computations".

`nsgapopulation.cpp` Contiene la implementación de la clase `nsgapopulation`. Esta se acopla sobre la clase `population` e implementa "nondominated sorting" y "crowding distance computation" requerido para NSGA-II.

`crossover.cpp` Contiene la implementación de la clase `crossover` y la implementación de los operadores de cruce.

`selection.cpp` Contiene la implementación de la clase `selection` y la implementación de los operadores de selección.

`localsearch.cpp` Contiene la implementación de la clase `localsearch` así como la implementación de un operador de búsqueda local.

`ga.cpp` Contiene la implementación de la clase `ga` y los bucles principales que corresponden a las diferentes arquitecturas de algoritmos genéticos GA y NSGA-II.

`random.cpp` Contiene las subrutinas relacionadas con el pseudo generador de números aleatorios.

`globalSetup.cpp` Contiene la implementación de la clase `globalSetup`, además de los datos globales necesitados para las diferentes clases.

`userDefinables.cpp` Contiene el código de la función objetivo, así como la función `main()` del programa. Para probar GAtbx en un problema en concreto, se modifica la función `globalEvaluate()` que se encuentra en este fichero.

En la siguiente figura (4.1) se detallan las dependencias que existe entre los ficheros, en base a los *includes* reflejados en los ficheros cabezera `.hpp`.

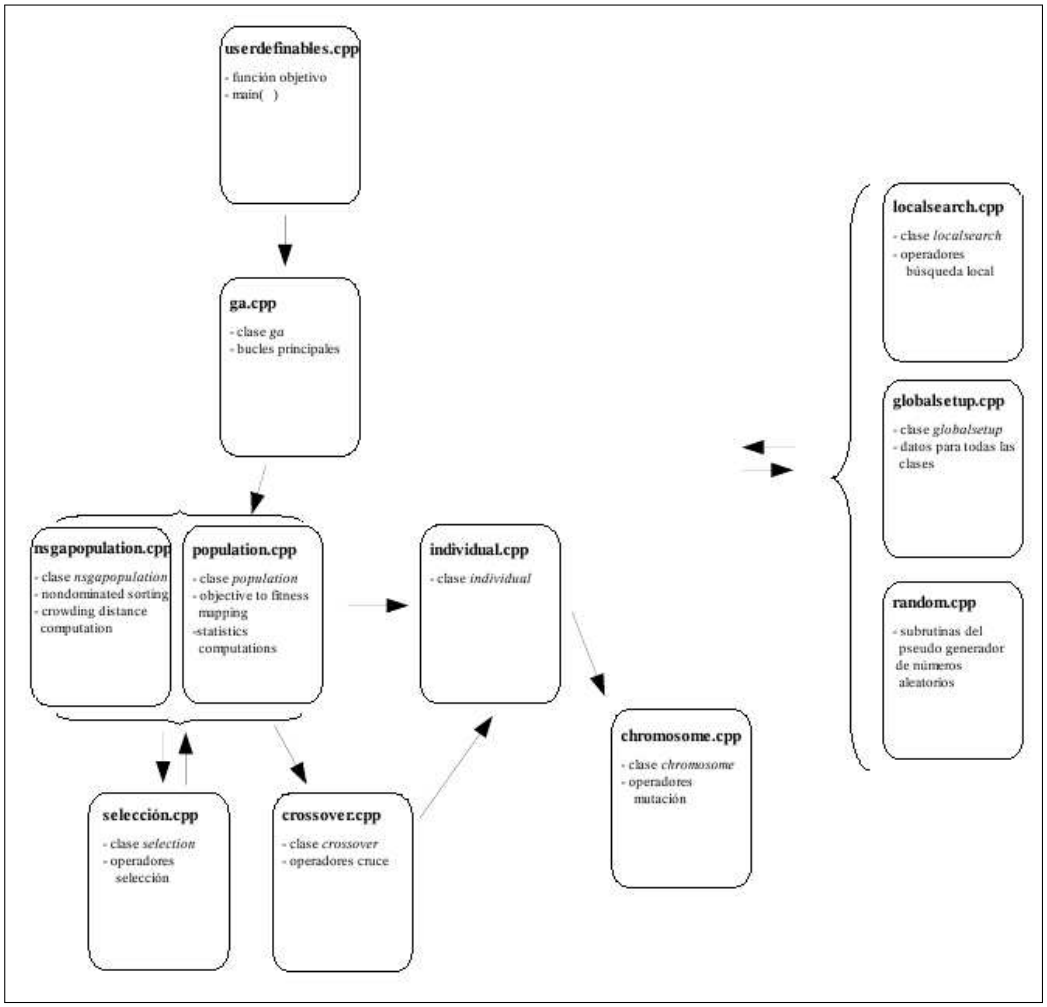

Figura 4.1: Dependencias entre ficheros. GA toolbox

4.2. Desarrollo del software escogido. MGA toolbox

En este apartado se tratará mostrar y describir los trabajos de adaptación del software de algoritmos genéticos GA toolbox, escogido en la sección 3.1. Estos, como se ha comentado, consisten en las adaptaciones para el AGS, la implementación del operador selección según la fórmula electoral escogida (ley d'Hondt) e implementación de las funciones entre otros. La nueva versión de este software, se denomina MGA toolbox (Modified Single and Multiobjective Genetic Algorithm Toolbox in C++) y aunque mantiene todas las opciones del software original, debido a estos "extras" y modificaciones, se ha decidido cambiarlo a este nombre. Otro de los motivos ha sido el de evitar confusiones entre el software original y el modificado. A continuación se mencionan estas modificaciones y extras:

- Discretización de la población inicial.
- Operadores cruce y mutación adaptados (binario).
- Nuevo operador selección basado en la ley d'Hondt.
- Implementación de funciones del PDEC-05 en el software.
- Selección de parámetros y salida de estos por fichero. Mostrar al finalizar el mejor individuo.

En cuanto a la compilación se realiza del mismo modo que para GA toolbox (consultar la documentación [Sas07]). Para la ejecución, es también igual (ver apartado 4.1.2), salvo que el ejecutable obtenido tras la compilación recibe el nombre de `MGAtbx`. Así la sinopsis para ejecutar el programa queda:

```
MGAtbx <input file name>
```

La relación de ficheros incluidos en este nuevo software se puede ver en el apartado 4.2.5. Además se recoge en este apartado la configuración de las opciones ofrecidas en la sección anterior 4.1 (en GA toolbox) junto con las nuevas opciones desarrolladas.

4.2.1. Adaptación al AGS

En las siguientes páginas se detallan las modificaciones realizadas para adaptar el software GA toolbox al AGS. Primeramente se respalda la elección del software de genéticos GA toolbox, pese a ser necesarias las adaptaciones. A continuación figura la codificación para cadenas de cromosomas propuesta en la bibliografía [Mic96] y las soluciones tomadas para la adaptación. Por último se muestra la implementación de tales soluciones.

4.2.1.1. Elección de GA toolbox

Como se vio, en la tabla 3.1 de la sección 3.1, eran necesarias algunas modificaciones sobre el código para poder adaptarlo totalmente al AGS. Pese a esto, la elección del software está respaldando por la fuente y a la multitud de sus opciones. En lo que respecta al software GA toolbox (ver sección 4.1) trabaja o con enteros o con reales y por lo tanto los operadores cruce y mutación, aunque si se tratan del "cruce simple" (o por un punto) y la mutación, no se realizan en binario. El AGS, cuando fue presentado por Goldberg [Gol89a], genera la población inicial en binario, eligiendo aleatoriamente unos o ceros y a partir de esta cadena de bits, realiza todas las operaciones (operadores), decodificando la cadena para las evaluaciones.

La explicación de que el software GA toolbox no trabaje en binario es que habitualmente, debido a las condiciones "generalistas" a las que van destinadas los paquetes de software para algoritmos genéticos, no se establece esta codificación. A cambio, lo que estos programas realizan es trabajar directamente sobre variables, y así, el punto de cruce (para el caso de cruce simple) se escoge sobre una cadena formada por valores (enteros, naturales o reales) en vez de bits y la mutación es realizada, por lo tanto, sobre una de estas variables. En otras palabras, como no se generan las variables codificadas, no se trabaja con bits, tal como se presenta en la figura 4.2. Con ello fomenta la "universalidad" ya que en muchos casos la codificación a escoger depende del problema y el programador (como se vio en el apartado 2.1.5).

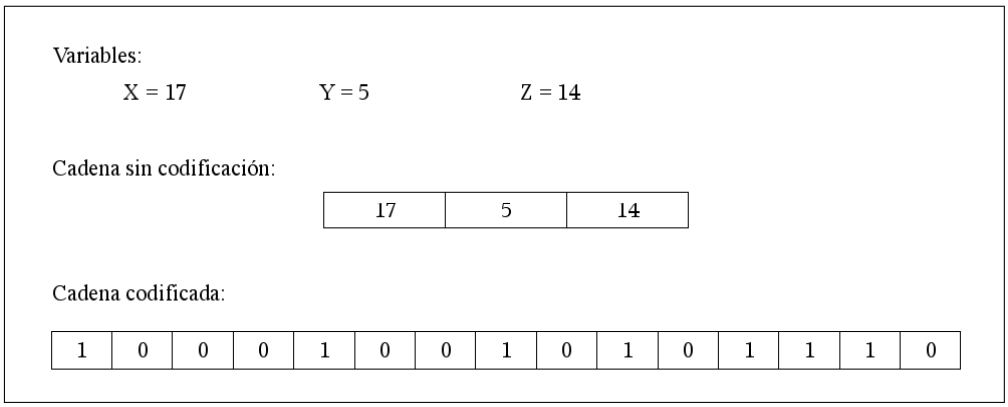

Figura 4.2: Ejemplo individuo sin codificar y codificado binario

4.2.1.2. Codificación

Los siguientes párrafos están extraídos del capítulo 2 *"GAs: How Do They Work?"* del libro de la bibliografía [Mic96]. La codificación propuesta en este libro es estándar y frecuentemente utilizada en el AGS que junto a la procedencia, ha sido la elegida para nuestro problema.

Supongamos que queremos optimizar una función de k variables, $f(x_1, \ldots, x_k) : R^k \rightarrow R$ y que cada variable x_i toma valores dentro del rango $D_i = [a_i, b_i] \subseteq \mathbb{R}$ y $f(x_1, \ldots, x_k) > 0$ para todo $x_i \in D_i$. Supongamos además cierta precisión requerida para los valores de las variables (por ejemplo $n = 6$ decimales).

Queda claro que para alcanzar esta precisión, cada dominio D_i, tiene que ser dividido en $(b_i - a_i) \cdot 10^n$ rangos de igual tamaño. Si definimos m_i el mínimo entero, tenemos que (4.3):

$$(b_i - a_i) \cdot 10^m \leq 2^{m_i} - 1. \tag{4.3}$$

Por lo tanto, la representación de cada variable x_i codificada como una cadena binaria de longitud m_i satisface claramente la precisión requerida. La siguiente expresión 4.4 permite obtener el valor x_i real de una determinada cadena binaria.

$$x_i = a_i + decimal(1001 \ldots 001_2) \cdot \frac{(b_i - a_i)}{2^{m_i} - 1} \tag{4.4}$$

Donde $decimal(string_2)$ representa el valor decimal de aquella cadena binaria.

Entonces, cada cromosoma es representado por una cadena binaria de longitud $m = \sum_{i=1}^{k} m_i$, donde m_1 corresponde a los valores del rango $[a_1, b_1]$, m_2 corresponde a los valores del rango $[a_2, b_2]$ y así sucesivamente hasta m_k corresponde a los valores del rango $[a_k, b_k]$.

A parte de estas definiciones procedentes del [Mic96], se estudió la operación opuesta, de la que se parte del valor real de x_i y se desea conocer la cadena binaria correspondiente. Esta vendría definida por la siguiente operación:

$$1001 \ldots 001_2 = binario\left(\frac{2^{m_i} - 1}{(b_i - a_i)} \cdot (x_i - a_i)\right) \tag{4.5}$$

A continuación se describe un ejemplo de representación. Partimos, de una función con dos variables ($k = 2$): $f(x_1, x_2) : R^2 \rightarrow R$. Cada variable x_i ($i = 2$) toma valores dentro de los siguientes rangos:

$$D_1 = [-20, 12] \subseteq \mathbb{R} \quad D_2 = [0, 32] \subseteq \mathbb{R} \tag{4.6}$$

La precisión requerida para los valores de las variables x_1 y x_2, será de 1 y 2 decimales respectivamente. Por lo tanto, ya está definido todo lo necesario para calcular el número de bits de cada variable y que serán los siguientes:

$$m_1 = \frac{log(((12 + 20) \cdot 10^1) + 1)}{log(2)} = 8{,}3264 \tag{4.7}$$

$$m_1 = \frac{log(((32 + 0) \cdot 10^2) + 1)}{log(2)} = 11{,}6443 \tag{4.8}$$

Lo que corresponde a 9 bits para x_1 y 12 para x_2. Si partimos de la siguiente cadena de caracteres 100010001010000100111, tendríamos que los primeros 9 bits pertenecen a x_1, que sería 100010001 y le corresponde el 273 en decimal, y los 12 siguientes pertenecerían a x_2, que sería 010000100111 y le corresponde el 1063 en decimal. Utilizando la expresión 4.4 descrita en este apartado, obtenemos los siguientes valores reales de nuestro ejemplo:

$$x_1 = -20 + 273 \cdot \frac{(12 + 20)}{2^9 - 1} = -2{,}9041096 \tag{4.9}$$

$$x_2 = 0 + 1063 \cdot \frac{(32 - 0)}{2^{12} - 1} = 8{,}3067155 \tag{4.10}$$

Como se puede ver, para los dos casos, el número de decimales que resultan de obtener el real, es mayor que el requerido. Esto es debido a que para los dos casos del ejemplo, ninguna de las dos cadenas de bits, coinciden con los valores reales del rango que cumplen que su producto con $\frac{(b_i - a_i)}{2^{m_i} - 1}$ (4.4) sea n veces (donde $\subseteq \mathbb{R}$). Recordamos que cada dominio D_i que es dividido en $(b_i - a_i) \cdot 10^n$ rangos de igual tamaño. El resto de reales no corresponde con el número de decimales requerido, sino que es mayor (asegura el requerido). En la tabla 4.1 se puede ver un ejemplo de ello.

4.2.1.3. Solución tomada

Para adaptar el GA toolbox al AGS, se ha procedido a la creación de variantes de los operadores "cruce por un punto" y "mutación de varios puntos" para hacer el cruce (simple) binario y la mutación en binario (unos y ceros). Estos consisten en buscar el punto de cruce (para el operador cruce) y el bit a permutar (para el operador mutación) en una cadena binaria, individuo, derivada de la codificación

de las variables del problema. Las nuevas opciones se incluyen, naturalmente, en MGA toolbox

El proceso a seguir es el de codificar los individuos (como el ejemplo de la figura 4.2), aplicar el operador (es decir, el cruce o la mutación) y por último descodificar y devolver el resultado. La siguiente figura 4.3 muestra un esquema del proceso.

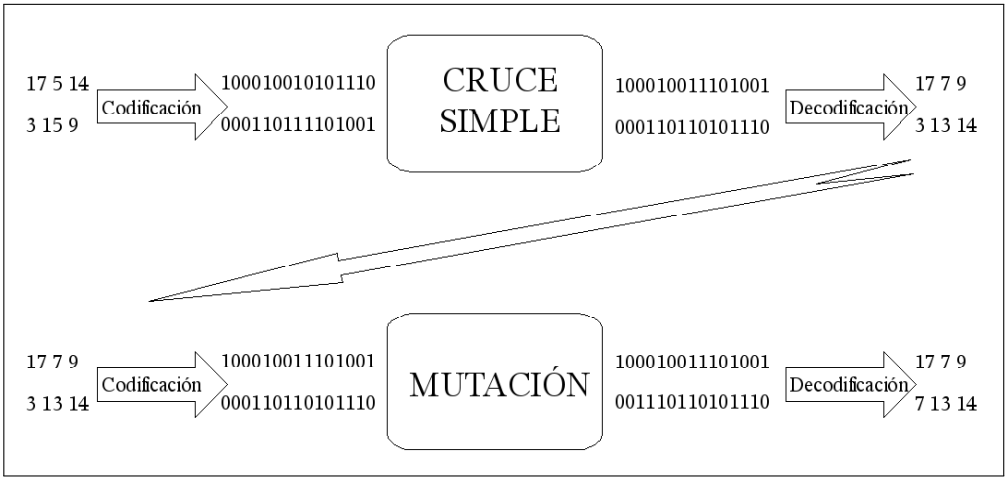

Figura 4.3: Esquema del nuevo proceso cruce/mutación (ejemplo)

Un fenómeno que no hay que perder de vista es que para la primera codificación, en la que se parte de la generación inicial, creada aleatoriamente de números reales, resulta altamente probable que los valores de x_i generados, no pertenezcan al conjunto de valores reales correspondientes a la codificación vista en este capítulo. Por ejemplo, supongamos que tenemos una variable x_1 toma valores dentro del rango $D_1 = [0, 1] \subseteq \mathbb{R}$ con un precisión requerida de $n = 1$ decimal. Si calculamos, según la expresión 4.3 el número de bits, obtendremos que será $m = 4$. Pues bien, en la siguiente tabla (4.1), se muestran las correspondencias entre los binarios y sus reales correspondientes.

Por lo tanto si se escoge un real al azar, por ejemplo el 0.432, no corresponde a ninguno de los de la tabla (4.1).

Para evitar este problema se ha decidió discretizar la población inicial, justamente después de que haya sido generada. Con el fin de que todos los rango fueran iguales, se discretiza hacia abajo dentro del rango $D_i' = [a_i, b_i + \dfrac{(b_i - a_i)}{2^{m_i} - 1}]$, en vez

Función	Precisión
0001	0.0
0001	0.06666666666666666667
0010	0.1333333333333333333
0011	0.2
0100	0.2666666666666666667
0101	0.3333333333333333333
0110	0.4
0111	0.4666666666666666667
1000	0.5333333333333333333
1001	0.6
1010	0.6666666666666666667
1011	0.7333333333333333333
1100	0.8
1101	0.8666666666666666667
1110	0.9333333333333333333
1111	1.0

Tabla 4.1: Ejemplo correspondencias binario/real

del $D_i = [a_i, b_i]$. Un ejemplo se puede ver en la figura 4.4, en el que se discretiza la función $f(x) = x$, donde $x \subseteq \mathbb{R}$ en el intervalo $[0, 10]$.

Recopilando esta discretización y los nuevos operadores, se muestra en la figura 4.5 el esquema general del nuevo AGS adaptado, partiendo de la figura 2.3 que se vio en el capítulo 2. Con los operadores de *nuevo cruce binario* y *nueva mutación binaria*.

4.2.1.4. Implementación de la discretización

La implementación de esta operación se realiza dentro del fichero `chromosome.cpp`, en el constructor por defecto `Chromosome()` de la clase `Chromosome`, lo que implica la acción cuando se creen el objeto (los cromosomas), es decir al principio del algoritmo.

Es importante saber que lógicamente solo habrá discretización cuando se haya seleccionado el cruce binario o la mutación binaria o bien los dos (ver nuevos

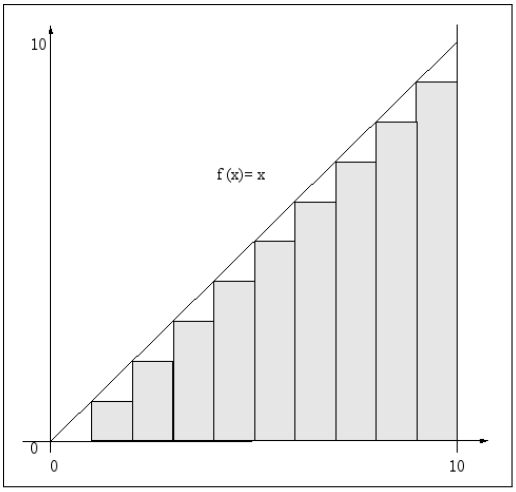

Figura 4.4: Ejemplo discetización hacia abajo.

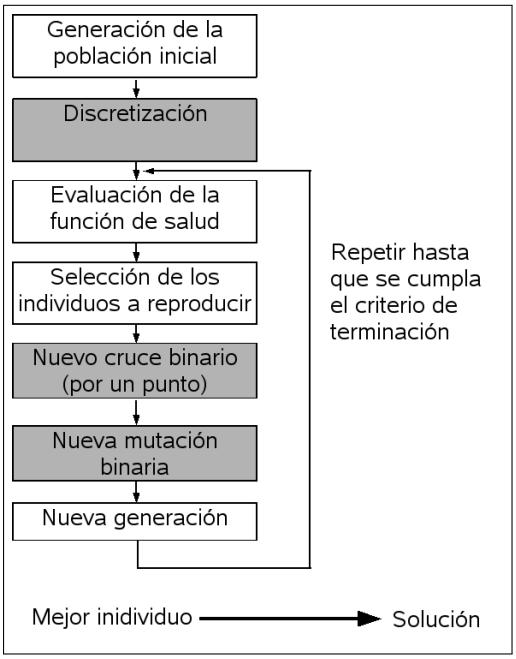

Figura 4.5: Esquema de funcionamiento del algoritmo genético adaptado (adaptado de [YBS+03])

operadores en apartado 4.2.1.5).

A continuación se muestra el código comentado de la discretización, explicada en este apartado y en la figura 4.4. Como se puede ver, se realiza un chequeo previo, de las opciones de cruce escogidas, para determinar si corresponde la mencionada discretización de la población inicial. El código para la discretización ha sido insertado en `chromosome.hpp`.

```
...
//check if the Crossover and Mutation type are binary
//In that case a discretización  o the initials values is needed

if((globalSetup->xOverType == BinaryOnePoint)|| (globalSetup->xOverType == BinaryTwoPoint)
     ||(globalSetup->mutationType == BinaryMut)){
 for (ii = 0; ii < globalSetup->noOfDecisionVariables; ii++) {
  pregenes=myRandom.boundedRandom(globalSetup->variableRanges[ii][0],
    (globalSetup->variableRanges[ii][1]+((globalSetup->variableRanges[ii][1]-
     globalSetup->variableRanges[ii][0])/(pow(2,binary.precision[ii])-1))));
  genes[ii]=((globalSetup->variableRanges[ii][1]-globalSetup->variableRanges[ii][0])
    /(pow(2,binary.precision[ii])-1))*(int(pregenes*(pow(2,binary.precision[ii])-1)
    /(globalSetup->variableRanges[ii][1]-globalSetup->variableRanges[ii][0])));
  }
}
else{
 for (ii = 0; ii < globalSetup->noOfDecisionVariables; ii++) {
  genes[ii] = myRandom.boundedRandom(globalSetup->variableRanges[ii][0],
  globalSetup->variableRanges[ii][1]);
  if (globalSetup->variableTypes[ii]==Integer)
     genes[ii] = int(genes[ii]+0.5);
  }
}
...
```

Se utiliza la variable `precision[ii]`, que indica la precisión a utilizar para la codificación, necesaria tal como se ha visto en la sección de codificación. Esta vendrá dada en el fichero de entrada del software y se cargará en esta variable.

4.2.1.5. Implementación de los nuevos operadores adaptados cruce y mutación

La implementación de estas variantes se realiza incluyendo nuevas clases y funciones en los ficheros `crossover` y `chromosome` que contienen las rutinas para este tipo de cruce y mutación, y por otro lado, se añade un nuevo fichero `binary.cpp` y su cabecera `binary.hpp` que contiene las rutinas complementarias para realizar la codificación y descodificación.

A continuación se muestra el código comentado correspondiente a los nuevos operadores cruce y mutación binarios cuyo proceso se ha comentado en este

apartado y en la figura 4.3. Consiste en codificar los individuos, según las expresiones vistas (consultar codificación, en especial 4.4 y 4.5), aplicar el operador (es decir, el cruce o la mutación) y por último descodificar para devolver el resultado.

Así, el código para el cruce binario insertado en la cabecera para el cruce `crossover.hpp` muestra la siguiente clase:

```
class OneTwoPointBinaryCrossover : public Crossover {
private:
  int noOfPoints;
public:
  OneTwoPointBinaryCrossover(int numPoints);
  void crossover (Individual *parent1, Individual *parent2);
};
```

Y su función miembro en el fichero `crossover.cpp` quedaría del siguiente modo:

```
/*
Single and Two Point Binary Crossover:
Single Point: Randomly select a crossover point between bits and exchange
the strings to the left of the crossover point to create two new individuals.
Two Point: Randomly select two crossover points between bits and exchange
strings between the two crossover points to creat two new indivduals.
*/
void OneTwoPointBinaryCrossover::crossover(Individual *parent1, Individual *parent2)
{
 int ii,jj, XoverPt1, XoverPt2;
 int Temp, start;
 Individual child1(parent1), child2(parent2);

 //Create needed arrays + allocation memory
 double *temp, *temp2;
 temp=new double[globalSetup->noOfDecisionVariables];
 temp2=new double[globalSetup->noOfDecisionVariables];
        //Create a new array to save the binary value of childs
 int *childBinary1,*childBinary2;
 childBinary1=new int[binary.totalLenght];
 childBinary2=new int[binary.totalLenght];

 int *childDecimal1,*childDecimal2;

//codificacion

 for(ii=0;ii<globalSetup->noOfDecisionVariables;ii++){
  temp[ii]=child1[ii];   temp2[ii]=child2[ii];;
 }

   // Call to the function codification (see binary.cpp)
   // An array is achieved and saved in childBinary:
   // all coded variables each individual are saved
   // together in this array

   binary.codification(temp,childBinary1);
```

```
   binary.codification(temp2,childBinary2);

//crossover process
 switch(noOfPoints) {

 //One point crossover
 case 1:
   XoverPt1 = myRandom.boundedIntegerRandom(0,(binary.totalLenght)-1);
   //Exchange the strings to the left of the crossover point
   //to create two new individuals
   for(ii = 0; ii <= XoverPt1; ii++) {
     Temp = childBinary1[ii];
     childBinary1[ii]=childBinary2[ii];
     childBinary2[ii]=Temp;
   }
 break;

 //Two points crossover
 case 2:
   XoverPt1 = myRandom.boundedIntegerRandom(0,(binary.totalLenght)-1);
   do {
     XoverPt2 = myRandom.boundedIntegerRandom(0,(binary.totalLenght)-1);
   }while(XoverPt1 == XoverPt2);
   if(XoverPt1 > XoverPt2) SWAP(XoverPt1, XoverPt2);
   // Exchange strings between the two crossover points to creat two new indivduals
   for(ii = XoverPt1; ii < XoverPt2; ii++) {
     Temp = childBinary1[ii];
     childBinary1[ii]=childBinary2[ii];
     childBinary2[ii]=Temp;
   }
   break;

 default: exit(0);
 }

//decodificacion:
 start=0; // needed to extract substrings

      for (ii=0;ii<globalSetup->noOfDecisionVariables;ii++){
      childDecimal1=new int[binary.precision[ii]];
      childDecimal2=new int[binary.precision[ii]];
      //extract substring (each substring belong to each variable)
      binary.subarray(childBinary1,childDecimal1,
                start,binary.precision[ii],binary.totalLenght);
      binary.subarray(childBinary2,childDecimal2,
                start,binary.precision[ii],binary.totalLenght);

      // Set the news individuals (decoded)
      child1.setValue(ii,globalSetup->variableRanges[ii][0]
        +(binary.Bin2Dec(childDecimal1,binary.precision[ii])
        *((globalSetup->variableRanges[ii][1]-globalSetup->variableRanges[ii][0])
        /(pow(2,binary.precision[ii])-1))));
      child2.setValue(ii,globalSetup->variableRanges[ii][0]
        +(binary.Bin2Dec(childDecimal2,binary.precision[ii])
        *((globalSetup->variableRanges[ii][1]-globalSetup->variableRanges[ii][0])
        /(pow(2,binary.precision[ii])-1))));

     start+=binary.precision[ii];
     delete[] childDecimal1;
     delete[] childDecimal2;
     }
    }
```

```
}

// free memory
  delete[] childBinary1;
  delete[] childBinary2;
  delete [] temp;
  delete [] temp2;

//If the variables are beyond the range then perturb it only till the range.
//In case of binary mutation is not necesary: BinaryMut will check it!
 if(globalSetup->mutationType != BinaryMut){
   for(ii = 0; ii < globalSetup->noOfDecisionVariables; ii++) {
     if(child1[ii] > globalSetup->variableRanges[ii][1]) child1[ii]
                                     = globalSetup->variableRanges[ii][1];
     if(child1[ii] < globalSetup->variableRanges[ii][0]) child1[ii]
                                     = globalSetup->variableRanges[ii][0];
     if(child2[ii] > globalSetup->variableRanges[ii][1]) child2[ii]
                                     = globalSetup->variableRanges[ii][1];
     if(child2[ii] < globalSetup->variableRanges[ii][0]) child2[ii]
                                     = globalSetup->variableRanges[ii][0];
  }
 }

// Create the descendants
 if(globalSetup->nichingType == DeterministicCrowding) {
   child1.evaluateFitness();
   child2.evaluateFitness();
   deterministicCrowding(parent1, parent2, &child1, &child2);
 }
 else {
   for(ii = 0; ii < globalSetup->noOfDecisionVariables; ii++) {
     parent1->setValue(ii, child1[ii]);
     parent2->setValue(ii, child2[ii]);
   }
 }
}
```

Como se puede comprobar, se ofrecen dos modalidades de cruce binario: simple o en un punto y múltiple o en dos puntos (sección 4.1).

En lo referente al código para el operador mutación binario, la función insertada en la cabecera para el cruce `chromosome.hpp` es la siguiente:

```
  void mutateBinary (const int *freezeMask);
```

Su implementación en el fichero `chromosome.cpp` quedaría del siguiente modo:

```
/*
 *============================================================
 * Function Name: mutateBinary
 * Function Task: Selective mutation. Randomly select one of
 *    the bits in the genes and permutate it.
 * Input parameters: None
 * Output: None
 *============================================================
```

```
 */

void Chromosome::mutateBinary (const int *freezeMask) {

 double temp;
 int ii, jj;
 //Create a new array to save the binary value of Genes
 int *bitGenes;

 //For each decision variable
 for (ii = 0; ii < globalSetup->noOfDecisionVariables; ii++) {
 //Allocation memory
  bitGenes=new int[binary.precision[ii]];
  temp=genes[ii];
      //codification (of the temporal variable)
     // Call to the function codification (see binary.cpp)

      binary.Dec2Bin(int(0.5+(((pow(2,binary.precision[ii])-1)/
       ((globalSetup->variableRanges[ii][1])-(globalSetup->variableRanges[ii][0])))
       *(temp-(globalSetup->variableRanges[ii][0])))),binary.precision[ii],bitGenes);

 // If the GA type is SGA then check if freeze mask of the variable is off
 if(((globalSetup->gaType == SGA)&&(freezeMask[ii] == OFF))||
                              (globalSetup->gaType == NSGA)) {

  //Flip a biased coin with mutation probability
  for (jj = 0; jj < binary.precision[ii]; jj++) {
    // Used a random Gaussian noise to chose a bit
    if(myRandom.flip(globalSetup->mutationProbability)) {
     // Spin the selected bit
      if(bitGenes[jj]>0){
       bitGenes[jj]=0;
      }
      else{
       bitGenes[jj]=1;
      }
     }
    }
  }

//decodificacion

     // Set the news individuals (decoded)
      genes[ii]= globalSetup->variableRanges[ii][0]+
                  (binary.Bin2Dec(bitGenes,binary.precision[ii])
         *((globalSetup->variableRanges[ii][1]-globalSetup->variableRanges[ii][0])/
          (pow(2,binary.precision[ii])-1)));

       //If the variables are beyond the range then perturb it only till the range
       if(genes[ii] > globalSetup->variableRanges[ii][1]) genes[ii] =
                         globalSetup->variableRanges[ii][1];
       if(genes[ii] < globalSetup->variableRanges[ii][0]) genes[ii] =
                         globalSetup->variableRanges[ii][0];


  //free memory
  delete [] bitGenes;
 }
}
```

Dentro del GA toolbox para el caso de la mutación polinomial, se incluye un chequeo de los "individuos mutados". Si estos nuevos individuos están fuera del rango definido para cada variable, se redondea al límite más próximo. Se ha evaluado esta opción encontrándola interesante para incorporarla a la mutación binaria, en el software MGA toolbox: *el proceso de mutación binaria adapta los valores que se salen del rango a los límites.*

Compatibilidades Debido a la naturaleza de estos nuevos operadores y la discretización se tuvo que tener en cuenta una serie de pautas para que no "pudieran mezclarse" con otros operadores y ocurrieran problemas en las ejecuciones. Las pautas son las siguientes:

1. Solamente se permitirán variables del tipo `double` con el cruce y la mutación binarios.

2. Siempre que se escoja cruce binario, debe de acompañarse con mutación binaria o viceversa.

```
for(ii = 0; ii < globalSetup->noOfDecisionVariables; ii++) {
 if((((globalSetup->xOverType == BinaryOnePoint) ||
  (globalSetup->xOverType == BinaryTwoPoint)) ||
   (globalSetup->mutationType == BinaryMut))&&
    (globalSetup->variableTypes[ii]== Integer)){
     printf(" Error! Binary cannot be used with integer variables\n");
     exit(1);
  }
 }

if(((globalSetup->xOverType == BinaryOnePoint) ||
  (globalSetup->xOverType == BinaryTwoPoint)) &&
     (globalSetup->mutationType != BinaryMut)) {
      printf(" Error! BinaryOne(Two)Point must be always used with BinaryMut\n");
    exit(1);}

 if((globalSetup->mutationType == BinaryMut) &&
  ((globalSetup->xOverType != BinaryOnePoint) &&
     (globalSetup->xOverType != BinaryTwoPoint))) {
      printf(" Error! BinaryMut must be always used with BinaryOne(Two)Point\n");
       exit(1);}
```

4.2.2. Creación de nuevos operadores

4.2.2.1. Programación del operador selección d'Hondt

Como ya se ha comentado al principio de este capítulo el objetivo principal consiste básicamente en comparar en el AGS el operador ruleta frente al nuevo operador selección basado en la fórmula electoral de divisores comunes escogida.

Uno de los aspectos más importantes para poder llevar a cabo esto, es disponer en el software elegido, GA toolbox, del nuevo operador selección basado en la ley d'Hondt (variante de las fórmulas electorales de divisores comunes escogida, consultar la sección 3.3). Sabemos que en el software, GA toolbox, el operador selección ruleta está disponible, pero evidentemente no lo es así para el nuevo operador. Por ello, se implementará este operador, que lo denominaremos **d'Hondt**, directamente en el software.

Para el nuevo operador se ha creado la clase *D_HondtLawSelection* derivada de la clase base *selection*. A continuación se muestra el código de esta clase en el fichero cabecera `selection.hpp`:

```
class D_HondtLawSelection: public Selection {
public:
   D_HondtLawSelection(const Population *pop);
   void select(int *matingPool);
 };
```

Como se puede ver está formada por un constructor y una función miembro de la clase. A continuación se muestra su implementación. Según lo programado, el constructor no hace nada y es en la función donde se implementa el operador selección d'Hondt en sí.

Constructor:

```
D_HondtLawSelection::D_HondtLawSelection (const Population *pop) :
  Selection(pop)
{
}
```

Función miembro (comentada):

```
...
/***
/***
D_Hondt Law Selection: an array sized equal as population size is filled
with the fitness value of each individual. The better individual (highest fitness)
is assigned as father for the next generation. Then in the array,
the fitness value of that individual is divided by n+1,
where n is times picked of that individual.
E.g. if the individual 2 was picked three times, his fitness value
in the array is equal as fitness  4. Repeat this process till N
individuals are selected.

Note: This selection scheme has been developed to test the PR (Part list)
d'Hondt method (http://en.wikipedia.org/wiki/D'Hondt_method) as mechanism
of selection in Genetic Algorithms. As the main feature,
this method favors large parties and coalitions over scattered small parties.
Note: D_HondtLaw selection should always be used with some scaling method.
*/
```

```
void D_HondtLawSelection::select(int *matingPool)
{
  int ii, kk, num, *winner, *n;
  double rndNo, num2, num3, *votes;
//Hidden threshold: any individual which does not receive
//this threshold (fitness value)
//will not have any participation in the selection process
  double sum,threshold;
  int *excluded,allexcluded;
  excluded=new int[globalSetup->populationSize];

  threshold=((double *)globalSetup->selectionParameters)[1];

//Calculate the threshold of the votes (fitness values) and exclude
the individual that no reach it

   for (ii=0;ii< globalSetup->populationSize;ii++){
      sum=pop->getFitness(ii)+sum;
      }

   for (ii=0;ii< globalSetup->populationSize;ii++){
      if(pop->getFitness(ii)<(sum)*(threshold))
         excluded[ii]=1;
      else
         excluded[ii]=0;
     }


//Check if all individuals are excluded (less than threshold)
//In that case, all the individuals will be parents (no selection)


    for (ii=0;ii< globalSetup->populationSize;ii++){
        if(excluded[ii]==0){
           allexcluded=0;
           break;
         }
       else
         allexcluded=1;
     }


if(allexcluded!=1){

//allocate memory
  winner = new int[globalSetup->populationSize];
  n = new int[(globalSetup->populationSize)];
  votes = new double[(globalSetup->populationSize)];

//Load and initialization of the array "votes"
//with the fitness of each individual
// and initialization of the array "n"
        for (ii=0;ii< globalSetup->populationSize;ii++){
                    if (excluded[ii]==0){
                        votes[ii]=pop->getFitness(ii);
                        n[ii]=0;
                        }
                    else{
                        votes[ii]=0.0;
                        n[ii]=0;
                        }
                }
```

```
//Pick the best individuals ( then the first individuals in the rank array)
//to be fathers
        for(ii = 0; ii < globalSetup->populationSize; ii++){
            winner[ii]=bestArray(votes);
            n[winner[ii]]+=1;
            votes[winner[ii]]=(pop->getFitness(winner[ii]))/(n[winner[ii]]+1);
        }

//Order randomly the values of the array winner
        myRandom.shuffleArray(winner, (globalSetup->populationSize)+1);

//Load the winners to the matingPool
  for(ii = 0; ii < globalSetup->populationSize; ii++){matingPool[ii]=winner[ii];
      }
//free memory
  delete []winner;
  delete []votes;
  delete []n;
  }

else{
  winner = new int[globalSetup->populationSize];
        for(ii = 0; ii < globalSetup->populationSize; ii++){
             winner[ii]=ii;
        }
        myRandom.shuffleArray(winner, (globalSetup->populationSize)+1);
        for(ii = 0; ii < globalSetup->populationSize; ii++){  matingPool[ii] = winner[ii];
         }
  delete []winner;
  }

}
...
```

A esta función se le pasa:

- Un puntero de tipo int (array dinámico) correspondiente a la población.
- Un puntero de tipo double correspondiente al umbral

Primeramente se procede a calcular la barrera electoral, teniendo en cuenta **función de salud de cada individuo** (lo que vendría a equivaler al número de votos alcanzados). Por tanto se chequea que individuos quedarían excluidos de la selección por no alcanzar el umbral mínimo. En caso de que todos quedasen excluidos, se ha decidido reordenarlos aleatoriamente y que pasen tal cual al siguiente operador (no hay selección), a fin de que un nuevo cruce obtenga individuos mejores.

En un primer intento de programar el d'Hondt, se enfocó el siguiente proceso que consiste en rellenar una matriz en forma de array doble, de $n\ x\ n$ elementos, con los distintos valores resultantes del cociente de la **función de salud de cada individuo** (lo que vendría a equivaler al número de votos alcanzados) por los

divisores (1-2-3-4, etc.) y su ordenación posterior (de mayor a menor). A partir de ahí se tomaba como primer candidato (a ser progenitor) el que corresponde al valor mayor y así sucesivamente hasta que se completa el conjunto de candidatos que corresponde exactamente al número de individuos de la población.

Sin embargo, el cálculo de este array doble, que se realizaba en cada iteración, en el operador de selección, consume muchos recursos informáticos, lo que hace que para altas poblaciones resultase un algoritmo muy lento. Por ejemplo, para una población de 200 individuos, teníamos 40000 elementos.

Como alternativa, se recurrió a utilizar un vector o array de una dimensión n: el tamaño de población. Primeramente este array es completado con los valores de salud de los respectivos individuos. A continuación se escoge el mejor individuo, que será el que mayor valor de salud tiene, y en el array se sustituye su valor de salud por la siguiente expresión 4.11. Este proceso se repite hasta que el conjunto de candidatos que corresponde exactamente al número de individuos de la población.

$$\text{valor de salud}/(\text{número de veces escogido} + 1) \tag{4.11}$$

A continuación se muestra un ejemplo del proceso que recorre el operador selección d'Hondt:

1. Carga del array de los cocientes con la función de salud de cada individuo (expresión 4.12). A continuación en la tabla 4.2 se pueden ver los valores de salud y sus cocientes.

Individuo	*fitness*	*fitness/1*	*fitness/2*	*fitness/3*	*fitness/4*
A	1.2901	1.2901	0.64505	0.430033	0.322525
B	1.26084	1.26084	0.63042	0.42028	0.31521
C	1.31446	1.31446	0.65723	0.438153	0.328615
D	0.104599	0	0	0	0

Tabla 4.2: Ejemplo d'Hondt. Valores de salud y sus cocientes

$$[1{,}2901(A), 1{,}26084(B), 1{,}31446(C), 0(D)] \tag{4.12}$$

   El individuo D queda excluido al obtener una cuota por debajo del umbral, marcado al 3 % (barrera electoral).

2. Selección del primer individuo que van a ser padre de la siguiente generación: C.

3. Sustituir el valor de salud de C por la expresión 4.11 (en este caso 2).

$$[1{,}2901(A), 1{,}26084(B), 0{,}65723(C), 0(D)] \tag{4.13}$$

4. Selección del segundo individuo que van a ser padre de la siguiente generación: A.

5. Sustituir el valor de salud de A por la expresión 4.11 (en este caso 2).

$$[0{,}64505(A), 1{,}26084(B), 0{,}65723(C), 0(D)] \tag{4.14}$$

6. Selección del tercer individuo que van a ser padre de la siguiente generación: B.

7. Sustituir el valor de salud de B por la expresión 4.11 (en este caso 2).

$$[0{,}64505(A), 0{,}63042(B), 0{,}65723(C), 0(D)] \tag{4.15}$$

8. Selección del cuarto individuo que van a ser padre de la siguiente generación: C.

9. Sustituir el valor de salud de B por la expresión 4.11 (en este caso 3).

$$[0{,}64505(A), 0{,}63042(B), 0{,}438153(C), 0(D)] \tag{4.16}$$

   Individuos seleccionados: C, A, B, C.

Como se puede ver, los mejores individuos de la población que tengan valores de función de salud muy superiores a los peores, pueden salir varias veces elegidos, ya que aunque este valor de la función de salud de los mejores fuera dividido por 1-2-3-4, etc. podría ser mayor al valor de salud dividido por 1 de los peores.

La función *bestArray* (objeto protegido dentro de la clase base, por lo tanto es accesible a las clases derivadas de la clase base) se ha implementado para obtener el mejor individuo del array. Esta función se comenta a continuación:

```
...
//This is a quick sort routine used for rank the array quation in
//the d'Hondt method. Return the array output ranked according the
//ranking of array input

int Selection::bestArray(double *array)
{
  int ii, best;
  best=0;
```

```
  for(ii=1;ii<globalSetup->populationSize;ii++){
    if(array[ii]>array[best]){ best=ii;
  }
  }
 return best;
}
...
```

Finalmente, reseñar que al igual que el software limita el operador "ruleta" para el tipo de de optimización SGA y no válido para NSGA (ver sección 4.1), igualmente queda el operador selección d'Hondt limitado a SGA.

4.2.3. Implementación de las funciones del PDEC-05

El conjunto de todas las técnicas y operaciones que figuran en esta memoria, carece de sentido si no se enfocan para la resolución de un problema de optimización. Para ello es necesario definir e implementar la función objetivo a optimizar. En el apartado 4.1.2 se indica dónde y cómo se debe insertar esta función a estudiar en el software GA toolbox, que para el nuevo software modificado, es exactamente igual.

En este proyecto, tal como se vio en el apartado D.1.0.6 del apéndice D, la test suite escogida PDEC-05 [SHL+05] contiene 25 funciones de las cuales, la función escogida en el apartado 3.2.2, deberá ser implementada en este software de algoritmos genéticos. En la dirección `http://www3.ntu.edu.sg/home/EPNSugan/` se puede descargar un programa al que devuelve el valor de cualquiera de las 25 funciones que completan la suite al introducir el número de variables y sus valores.

La naturaleza de estas funciones (a las cuales se les introduce ruido, son rotadas y desplazadas...) dificulta ser escritas del mismo modo que se indica en el apartado 4.1.2, es decir de un modo directo. Partiendo que se dispone de este programa con las funciones, se decidió "acoplarlo" al MGA toolbox.

La implementación de las 25 funciones del PDEC-05 se realiza "traspasando y adaptando" el código en dos ficheros C++, `objfunc.cpp` y `functions.cpp`, y su respectivos ficheros de cabecera, `obfunc.hpp` y `functions.hpp`, quedando del siguiente modo:

`objfunc.cpp` Contiene las rutinas que componen y controlan cada función, rotaciones, desplazamientos, ruido... así como las inicializaciones y reserva de memoria.

`functions.cpp` Contiene la implementación de las funciones básicas (Rastrigin, Schwefel, Esfera...) que se utilizan para formar las funciones más simples y a las compuestas del PDEC-05.

Con ellos, según se indica en el apartado 4.1.2 la llamada a la función objetivo se realiza en el fichero `userDefinables.cpp` por medio de la función:

```
double suganthan_benchmark_func(double *x, int noOfDecisionVariables, int function);
```

A esta función se le pasan:

- Un array x con el valor de las variables.
- Un entero correspondiente al número de variables de decisión.
- Un entero (entre 1 y 25) correspondiente al número de la función del PDEC-05 que deseemos utilizar.

Abajo se muestra como se usaría la función descrita en el fichero `userDefinables.cpp`, para la función 1.

```
...
/* Objective function and constraints go here */
void globalEvaluate(double *x, double *objArray,
double *constraintViolation, double *penalty, int *noOfViolations)
{
   int ii;
   FILE *outEvals;
   FILE *outResults;

   for(ii = 0; ii < globalSetup->finalNoOfObjectives; ii++)
     objArray[ii] = 0.0;

   if(globalSetup->gaType == SGA) {

     *objArray = myobjfunc.suganthan_benchmark_func(x,
globalSetup->noOfDecisionVariables,1);

   }
...
```

Debido a que algunas de las rutinas de los ficheros `objfunc.cpp` y `functions.cpp` mencionados exigen ciertas funciones de generación de valores aleatorios, se han añadido las correspondientes nuevas rutinas al fichero `random.cpp`.

Por último, cada función lee datos de un fichero de datos asociado (se encuentran en el software descargable del PDEC-05), que contiene las matrices de transformación lineal (para los giros), los vectores de desplazamiento y

otras matrices necesarias para las funciones. Estos ficheros se han ubicado dentro de la carpeta `input_data` en el directorio del software MGA toolbox. Así, por ejemplo, la llamada por parte de la función Esfera a su fichero asociado `sphere_func_data.txt` se realiza del siguiente modo, dentro del `objfunc.cpp`:

```
 ...
FILE *fpt;
   fpt = fopen("input_data/sphere_func_data.txt", r");
      if (fpt==NULL)
        {
            fprintf(stderr,"\n Error: Cannot open input file for reading \n");
            exit(0);
        }
        ...
```

4.2.4. Otras modificaciones

A continuación se muestran modificaciones adicionales que también han sido necesarias, con el fin de mejorar algún aspecto o poder utilizar el software de tratamiento de datos que veremos posteriormente. Por no complicar demasiado la modificación o por no ser posible, se ha configurado para que esté disponible solamente para SGA y no anidamiento (ver sección 4.1, tipo de optimización y anidamiento).

4.2.4.1. Definición de los parámetros de salida a fichero

En la próxima sección 4.3 se tratarán los resultados devueltos por este software (MGA toolbox) a través de otro programa a través de un fichero. Para que este programa de tratamiento de datos pueda funcionar, se han escogido una serie de parámetros "útiles" o necesarios de la población/individuos al final de la ejecución y en cada iteración para poder llevar a cabo los 5 criterios del procedimiento de evaluación (sección 4.3). Basándonos en que este fichero debe ser leído por otro programa, para que esto resulte lo más sencillo posible, se ha prestado especial atención a la colocación de los parámetros. La imagen 4.6 muestra un esquema que relaciona el fichero con el software MGA toolbox y posibles programas que podrían tomar datos de este.

El programa GA toolbox ya consta de una opción para salvar resultados, más bien salva los individuos de la población. Con el fin de no alterar demasiado el código y mantener esta opción para MGA toolbox, no se ha modificado, sino que se ha añadido una nueva opción para salvar resultados (algo distinto a salvar individuos) en un fichero. Esta nueva opción se denomina *SaveResults*. El código añadido en

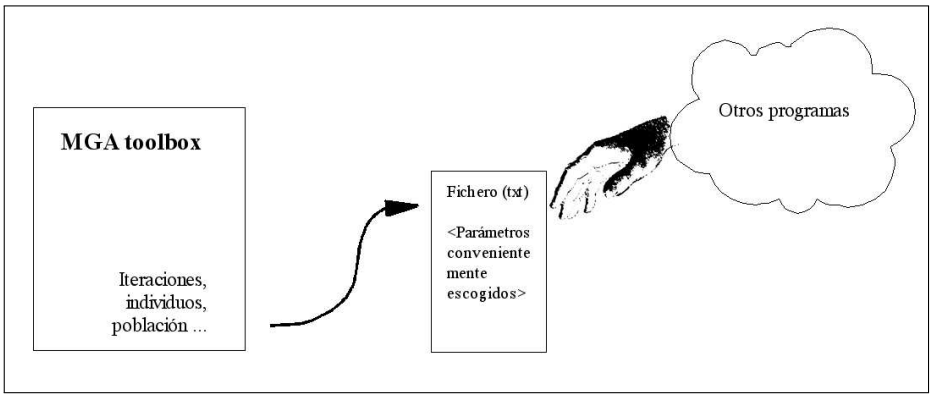

Figura 4.6: Esquema del fichero/programas

el fichero `userDefinables.cpp` para conseguir esta opción, crea el fichero con el nombre que se designe, se muestra a continuación:

```
// Save results

  //Code to read the information from the inputfile:
 if ((pToken = readOneLine(caBuf, ciBufSize, fInput)) == NULL) {
    fclose(fInput);
    printf(" Error in the input file, please refer to the documentation\n");
    exit(1);
  }
  if(strcmp("default", pToken) == 0) {
    printf("Saving the evaluated individuals to a file by default.\n");
    globalSetup->saveResults = true;
    globalSetup->saveEvalResults = new char[256];
    printf(.Enter the filename you want to save the population to.\n");
    fflush(stdout);
    scanf("%s", globalSetup->saveEvalResults);
  }
  // Check if the save results is 0 or 1
  else {
    fflush(stdout);
    // Save the results to a file
    if(atoi(pToken) == 1) {
      globalSetup->saveResults = true;
      // Filename to save the evaluated solutions to.
      globalSetup->saveEvalResults = new char[256];
      if ((pToken = strtok(NULL, BLANK_STR)) == NULL) {
fclose(fInput);
printf(" Error in the input file, please refer to the documentation\n");
exit(1);
      }
     //Save the filename in an array
      strcpy(globalSetup->saveEvalResults, pToken);
    }
  }
  fclose(fInput);
  fflush(stdout);
```

```
  if (globalSetup->gaType==SGA) {
    if(globalSetup->nichingType == NoNiching) {
          if(globalSetup->saveEvalResults) {
           //Create a new file to save it
           fOutput = fopen(globalSetup->saveEvalResults, "w");

           //First Headlines
           fprintf(fOutput, "FES\t");
           fprintf(fOutput, "population size\n");

           //save FES only for NoOfEvaluations. In other case save 0 FES
           if (globalSetup->noOfStoppingCriterias==1){
              for(ii = 0; ii < globalSetup->noOfStoppingCriterias; ii++){
                 switch(globalSetup->otherStoppingCriteria[ii]) {
                   case NoOfEvaluations:
                      fprintf(fOutput,"%f\t", globalSetup->stoppingParameter[ii]);
                      break;
                   default:
                      fprintf(fOutput,"0\t");
                      exit(0);
                   }
                 }
            }
           else{
                 fprintf(fOutput,"0\t");
           }

           //save the population size
           fprintf(fOutput,"%d\n", globalSetup->populationSize);

           //Second Headlines
           fprintf(fOutput, "Generation\t");
           fprintf(fOutput, "Maximum Fitness\t");
           fprintf(fOutput, "Best Objective Function value\n");
          }

    fclose(fOutput);
  }
}

  else{
printf("Save the results of the runned program to a file\n");
printf("is only for SGA and no niching available\n");
        exit(1);
  }
```

Además esta parte de código, como se puede ver, realiza adicionalmente las siguientes operaciones:

- Crea los primeros encabezados:
  - FES: número de evaluaciones de la función objetivo
  - Population size: tamaño de la población
- Solamente compatible para el criterio de finalización `NoOfEvaluations`, se salva el valor de "FES para parar el algoritmo". FES es el parámetro de este criterio de finalización que se inserta en el fichero de entrada de MGA toolbox. En caso de que se haya escogido otro criterio de finalización.
- Crear los segundos encabezados:
  - Generation
  - Maximum Fitness: máximo valor de salud en la población (en cada generación)
  - Best Objective Function value: mejor valor de la función objetivo en cada generación

Por lo tanto, estos son los parámetros escogidos para que posteriormente el programa de tratamiento de datos pueda elaborar los 5 criterios escogidos del proceso de evaluación.

Del fichero `ga.cpp` se recoge la generación que se está procesando, del fichero `population.cpp` el máximo valor de salud y por último, del fichero `individual.cpp` el valor de la función objetivo. A continuación se muestra el código.

Código para la generación:

```
...
FILE *outResults;
float decimal;
if(globalSetup->saveResults) {
     if (globalSetup->gaType==SGA) {
        if(globalSetup->nichingType == NoNiching) {
           outResults = fopen(globalSetup->saveEvalResults, .a");
           if (genID>0) fprintf(outResults, "\n", genID);
           fprintf(outResults, "%d\t", genID);
           fflush(outResults);
           fclose(outResults);
        }
     }
    }

...
```

Código para el máximo valor de salud:

```
...
     if(globalSetup->saveResults) {
         outResults = fopen(globalSetup->saveEvalResults, .a");
         fprintf(outResults, "%f\t", *(pop.maxfit));
         fflush(outResults);
         fclose(outResults);
   }
..
```

Código para el valor de la función objetivo:

```
...
       if(globalSetup->saveResults) {
           if (globalSetup->gaType==SGA) {
             if(globalSetup->nichingType == NoNiching) {
                outResults = fopen(globalSetup->saveEvalResults, .a");
                fprintf(outResults, "%.8f\t", *(pop.bestobj));
                fflush(outResults);
                fclose(outResults)
             }
           }
        }
...
```

Como se puede ver, cuando se imprime al fichero el valor de la función objetivo, se impone que este se realice con 8 decimales. Esto es debido a que como se verá en la sección 4.3, una de las premisas, era la de finalización, que establecía dos criterios: antes de alcanzar el número máximo de evaluaciones o cuando se de que el error en la función evaluada sea menor que 10^{-8} (es decir; Ter_Err o Max_FES). Por lo tanto, basándonos en este último criterio, es evidente que como mínimo se debe imprimir un valor con 8 decimales.

4.2.4.2. Mostrar mejor individuo

Aunque GA toolbox muestra en cada iteración el mejor individuo, una de las "carencias" más importantes que mostraba este software era que después de las "n" iteraciones, no mostrara el mejor individuo de todas ellas. Partiendo de que el objetivo de los algoritmos genéticos es el de encontrar el óptimo, o en su defecto la mejor solución posible tras la ejecución, se hace indispensable obtener este "mejor individuo de todas las generaciones". Esto es debido a que en la propia evolución de la población, puede ser que el mejor individuo empeore (ya sea por cruce o por mutación) y se pierda esa solución.

La solución que se ha tomado es ver para cada iteración si el mejor individuo de esa generación es peor o mejor que el mejor de las anteriores generaciones, el cual habrá sido guardado previamente. Entonces:

- En caso de que sea mejor, este nuevo individuo se guarda y sustituye al mejor de las anteriores generaciones.
- En caso de que sea peor, sigue quedando guardado el individuo mejor.

Por supuesto que el "mejor individuo de todas las generaciones" *no participa* en la población/evolución, es decir, en la selección, cruce y mutación. Es simplemente guardarlo como un "record guinnes" (por ejemplo, el hombre más alto existido nunca).

Esta función solo está disponible para el tipo de optimización SGA (como ya se ha comentado), debido a que es el tipo utilizado para este proyecto y que su naturaleza hace que sea más sencillo e intuitivo de implementarlo.

Primeramente se mostrará el código implementado para crear y guardar el mejor individuo de todas las generaciones. Esto se efectuará dentro de `population.hpp` y `population.cpp`.

Se realiza la definición del nuevo objeto `*allbestInd` perteneciente a la clase `Individual`. Este objeto es variable miembro en la sección privada de la clase `Population`.

```
...
  Individual *allbestInd;         the best individual of all runned generations
...
```

El nuevo objeto se creará dentro del constructor de población, en `population.cpp`.

```
Population::Population() {
...
  allbestInd= new Individual;
...
}
```

Una vez definido y creado, durante la ejecución del programa, tal como se ha comentado en este apartado, si el mejor individuo de esa generación `bestInd` es mejor que el mejor de las anteriores generaciones `allbestInd`, el nuevo individuo se guarda y sustituye al mejor de las anteriores generaciones.

```
/***
Replace the old population with the new population.
Incorporates elitism using the parameter globalSetup::replaceProportion.
If the value of replaceProportion is less than 1.0
then the bottom (100*replaceProportion)% individuals from the current population
are replaced by the top (100*replaceProportion)% individuals
from the new population.
Replace the best individual of all generations before for the best individual
```

```
of the current generation , if it is better.
*/
void Population::replacePopulation(void)
{
...
    if(selection->betterIndividual(bestInd, allbestInd)) allbestInd=bestInd;
...
}
```

Por último, una vez finalizado el programa, se mostrará el "mejor individuo de todas las generaciones". Esto se realiza por medio de una función miembro `void print_all_best_guy(void)` de la clase `Population`, que imprime por pantalla el este individuo. Si se ha escogido la opción de salvar la población en un fichero, al final de ese fichero también figurará este individuo.

La llamada a esta función se realiza en `ga.cpp` dentro de la función `GA:generate()` (función que implementa el bucle principal del algoritmo genético).

```
/***
Message EXIT_SUCCESS at the end of the program
Print the best individual
*/
void Population::print_all_best_guy(void){
    ofstream f2;
      cout <<std::endl<<std::endl<<"*********************************"<<std::endl;
      cout <<."Executed Programm without error"<<std::endl;
      cout <<"Best all individual"<<std::endl<<allbestInd[0]<<std::endl;

  if(globalSetup->savePopulation) {
      f2.open(globalSetup->saveEvalSolutions, ofstream::out| ofstream::app);
      f2 <<std::endl<<std::endl<<"*********************************"<<std::endl;
      f2 <<"Best all individual"<<std::endl<<allbestInd[0]<<std::endl;
      f2.close();
     }
}
```

4.2.5. Paquete software resultante: visión general

Tras los cambios añadidos y ficheros al software original, GA toolbox, se muestra la dependencia entre los ficheros del nuevo software, MGA toolbox, en base a los *includes* reflejados en los ficheros cabecera `.hpp` queda según se muestra en la siguiente figura 4.7

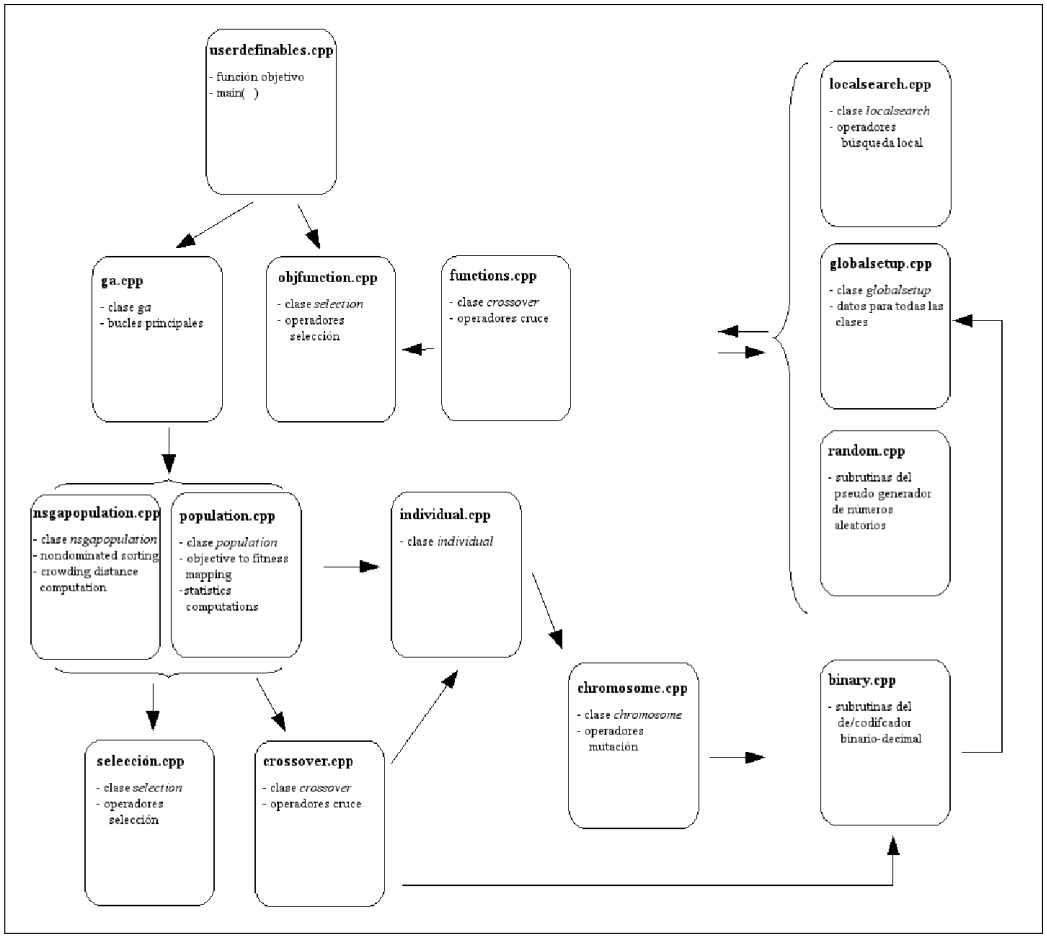

Figura 4.7: Dependencia entre ficheros. MGA toolbox

Con las modificaciones presentadas en esta sección podemos decir que hemos alcanzado el objetivo instrumental 4, al permitirse la ejecución del algoritmo genético simple en el software de computación evolutiva escogido. Por otro lado, gracias al resto de modificaciones, podemos decir que GA toolbox, ahora MGA toolbox, queda preparado para la siguiente fase, la experimentación.

4.3. Programa de análisis de resultados según PDEC-05: Statistics

Para procesar los 4 primeros criterios de evaluación del PDEC-05 (el 5º, Algorithm Complexity, se tratará aparte), se ha desarrollado un software aparte de MGA toolbox. Remitiéndonos a la figura 4.6 del apartado 4.2.4.1, este programa, Statistics, recoge los resultados volcados a fichero por el algoritmo genético. Por ello la mencionada figura queda como la siguiente 4.8. Acorde con el test PDEC-05 realiza un tratamiento de estos mostrando los resultados y estadísticas de la evaluación/evaluaciones.

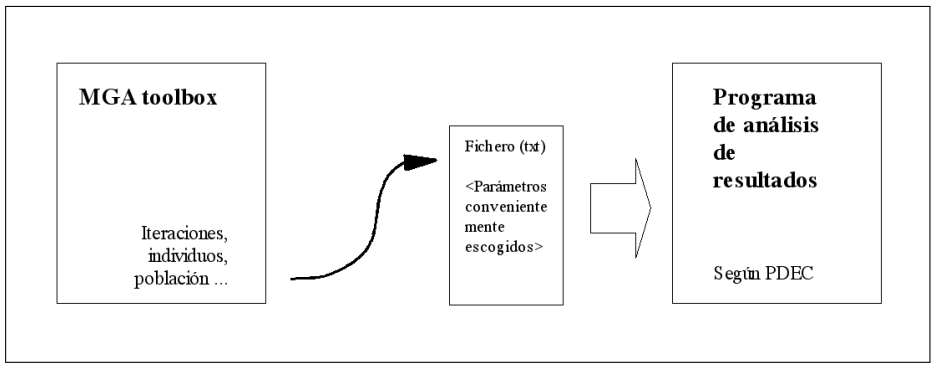

Figura 4.8: Esquema del fichero-programa Statistics

A continuación se recuerdan los 5 criterios:

1. **Salvar el valor de error de la función** $(f(x) - f(x^*))$ **después de** $1 \cdot 10^3$**,** $1 \cdot 10^4$ **y** $1 \cdot 10^5$ FES **(evaluaciones de la función objetivo) y en la finalización para cada ejecución (dada por** Ter_Err **o** Max_FES**).**

   Ordenar desde el menor al mayor el valor de error de las 25 ejecuciones.

   Presentar: el 1º (el mejor), el 7º, el 13º , el 19º y el 25º (el peor) valor de función, la media y la desviación estándar para las 25 ejecuciones.

2. **Salvar el número máximo de evaluaciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión.**

   El nivel de precisión para de la función escogida f_1, (función Esfera desplazada, apartado 3.2.2) es de -450+1e-6.

   Ejecución con éxito: Ejecución durante la cual, el algoritmo alcanza el nivel de precisión fijado dentro del Max_FES para una dimensión en particular.

Para cada función /dimensión, ordenar FES en las 25 ejecuciones desde el más pequeño (el mejor) hasta el mayor (el peor).

Presentar: el 1º (el mejor), el 7º, el 13º , el 19º y el 25º (el peor) valor de función, la media y la desviación estándar para las 25 ejecuciones.

3. **Ratio de éxito y rendimiento de éxito para cada problema**

   El ratio con éxito, acorde con el nivel de precisión, se define:

$$\text{Ratio de éxito} = \frac{\text{nº de ejecuciones con éxito}}{\text{nº total ejecuciones}}; \tag{4.17}$$

   El ratio de rendimiento de éxito se define:

$$\text{Rdto. de éxito} = \frac{(\text{FES para ejecución con éxito}) \cdot (\text{nº total ejecuciones})}{\text{nº total ejecuciones}}; \tag{4.18}$$

   Ambos ratios se calculan para cada función por separado.

4. **Gráficas de convergencia**

   Se realizan para cada problema pero solamente para $D = 30$. La gráfica muestra el rendimiento medio del total de ejecuciones finalizadas (Max_FES o Ter_Err). En el eje de ordenadas se representa $log_{10} = f(x) - f(x^*)$) y en el de coordenadas el FES (para cada función).

5. **Algorithm Complexity**

   Consta de los siguientes pasos:

   - Ejecutar el siguiente programa:

```
for i=1:1000000
x= (double) 5.55;
x=x + x; x=x./2; x=x*x; x=sqrt(x); x=ln(x); x=exp(x); y=x/x;
end
```

     El tiempo de ejecución de este se define como T_0

   - Obtener el tiempo de ejecución de 200000 evaluaciones (solamente evaluaciones, no con el resto de algoritmo) de la función 4 para una determinada dimensión D. Esto nos da T_1.

   - Tiempo de ejecución completo del algoritmo para la función 4 con 200000 evaluaciones y la misma dimensión D, que corresponde a T_2. Ejecutar este paso 5 veces para calcular $\hat{T_2} = mean(T_2)$.

Una vez obtenidos estos tiempos se define el Algorithm Complexity como:

$$\text{Algorithm Complexity} = \frac{\hat{T}_2 - T_1}{T_0}; \tag{4.19}$$

El Algorithm Complexity debe ser calculado para $D = 10$, $D = 30$ y $D = 50$, con el fin de mostrar la relación con la dimensión. Este criterio nos muestra la complejidad del algoritmo, independizándolo de la potencia de la máquina donde se corre, ya que toma referencias de tiempos (en una máquina) y los relaciona con los de la ejecución del algoritmo (en esa misma máquina).

Primeramente pareciera la mejor opción procesar todos estos puntos en el software del algoritmo genético MGA toolbox, pero se decidió procesar a parte por los siguientes motivos:

- Evitar insertar más líneas de código y ficheros ajenos al MGA toolbox, ya que podría aumentar la probabilidad de bugs en el software del algoritmo genético y le restarían flexibilidad (es decir poder usarlo para otros procedimientos de evaluación)

- El procedimiento de evaluación exige almacenar datos en cada ejecución (hasta 25) y una vez se recojan estos datos, realizar estadísticas. Por lo tanto es necesario que haya un programa que calcule estadísticas al final de las 25 ejecuciones, que es necesariamente un programa externo.

Modificaciones del procedimiento de evaluación Debido a que el objetivo principal del proyecto (consultar la sección 1.2, objetivos) no persigue exactamente los mismos fines que el PDEC-05, se han introducido dos pequeñas variaciones. Así resulta más beneficioso para la experimentación, se amolda a los objetivos y no queda tan enfocado a la "competición" para la que se creó. Estas modificaciones se enumeran a continuación:

- Gráficas de convergencia: según el PDEC-05 estás serán exclusivamente para $D = 30$. Para nuestro caso se representarán también para $D = 10$ y $D = 50$. Además no solamente se va a representar la media, sino que también se representarán las 25 ejecuciones, ya que en nuestro caso solamente se estudia una función (objetivo principal).

- Se añaden nuevos niveles de precisión para cada función, menos restrictivos, para el caso de que el algoritmo no llega nunca al nivel de precisión del

PDEC-05 marcado. El fin es que el número máximo de evaluaciones necesitado en cada ejecución para alcanzar la precisión, no salga siempre igual a Max_FES, que el ratio de éxito no salga siempre igual al 0 y que el ratio de rendimiento de éxito igual $indt$. Con ello, nos podrán resultar útiles estos criterios para nuestros propósitos comparativos. Los distintos niveles de precisión se determinarán según se comporte el algoritmo (ver capítulo 5).

4.3.1. Statistics. Programa en C++

En un principio, para implementar Statistics, se pensó en almacenar los datos en una base tipo access y programar con basic. Enseguida se vio que no era el lenguaje más idóneo que C++ debería ser el lenguaje en el que se debería implementar este programa. Algunos de los motivos fueron:

- Dificultades para leer los datos generados por el programa principal y las incompatibilidades surgidas. El programa además tardaba mucho en cargar estos datos (¡hasta 250000 filas de datos!)[3].
- Una vez cargados era necesario hacer programación en basic. La programación en basic es más intuitiva que en C++, pero al final no evitábamos crear líneas de código.
- El programa en C++ es más eficiente y directo que access+basic: ahorro de tiempo en la fase de experimentación. Además es altamente portable.
- Para el quinto punto de la evaluación (Algorithm complexitiy), es necesario realizar un par de programas en el mismo código que en el algoritmo inicial (implementado en C++).

4.3.2. GNUplot

Gnuplot[4] [GNU09] es un programa muy flexible para generar gráficas de funciones y datos. Este programa se ha escogido para la representación de las gráficas de convergencia que el PDEC-05 nos pide. Será ejecutado desde el programa statistics.

Gnuplot puede producir sus resultados directamente en pantalla, así como en multitud de formatos de imagen, como PNG, EPS, SVG, JPEG, etc. Se puede usar interactivamente o usando scripts (fichero de instrucciones). Este programa tiene

[3]También se estudió utilizar Excel por su capacidad de cálculo, pero debido a su limitación de filas, se desestimó

[4]Información y descargas (gratuitas) en: http://www.gnuplot.info/

gran base de usuarios y está convenientemente mantenido por sus desarrolladores. Entre su ventajas tenemos que por un lado existe una ingente cantidad de ayuda en Internet y por otro lado que es compatible con los sistemas operativos más populares (Linux, UNIX, Windows, Mac OS X...).

4.3.3. Compilación del programa

En el archivo de cabecera `statistics.hpp` es necesario "descomentar" la línea `# define f*` correspondiente a la función con la que se esté trabajando, antes de compilar el programa.

Las 25 funciones disponibles son las correspondientes al PDEC-05 y se pueden consultar en el apartado D.3.3 del apéndice D.

En Linux El código en Linux se ha compilado desde terminal de línea de comandos con el compilador GNU C++[5]. Además de tener instalado este compilador será necesario haber instalado también gnuplot (para instalación consultar apartado 4.3.2).

Gracias al archivo `makefile` que contiene las órdenes de compilación, solamente será necesario teclear en el terminal *"make"* (sin comillas) para obtener los ejecutables del programa. Estos ejecutables son: `test_1_run` y `test_25_runs`.

En Windows Para compilar el código se necesita tener un compilador de C++ instalado y seguir los pasos habituales para ese compilador. Al igual que para Linux será necesario haber instalado también gnuplot (para instalación consultar apartado 4.3.2).

4.3.4. Ejecución y funcionamiento del programa

Como se ha comentado en el apartado anterior, en la compilación se generan dos ejecutables: `test_1_run` y `test_25_runs`. El fin es ejecutar `test_1_run` cada una de las 25 veces que se ejecuta el algoritmo genético para coleccionar datos y estadísticas de cada una de estas. Entonces, al final de las 25 ejecuciones, se utiliza `test_25_runs` para que reúna y obtenga las estadísticas del procedimiento de evaluación (criterios).

[5]Descargable gratuitamente en: http://www.gnu.org/software/gcc/

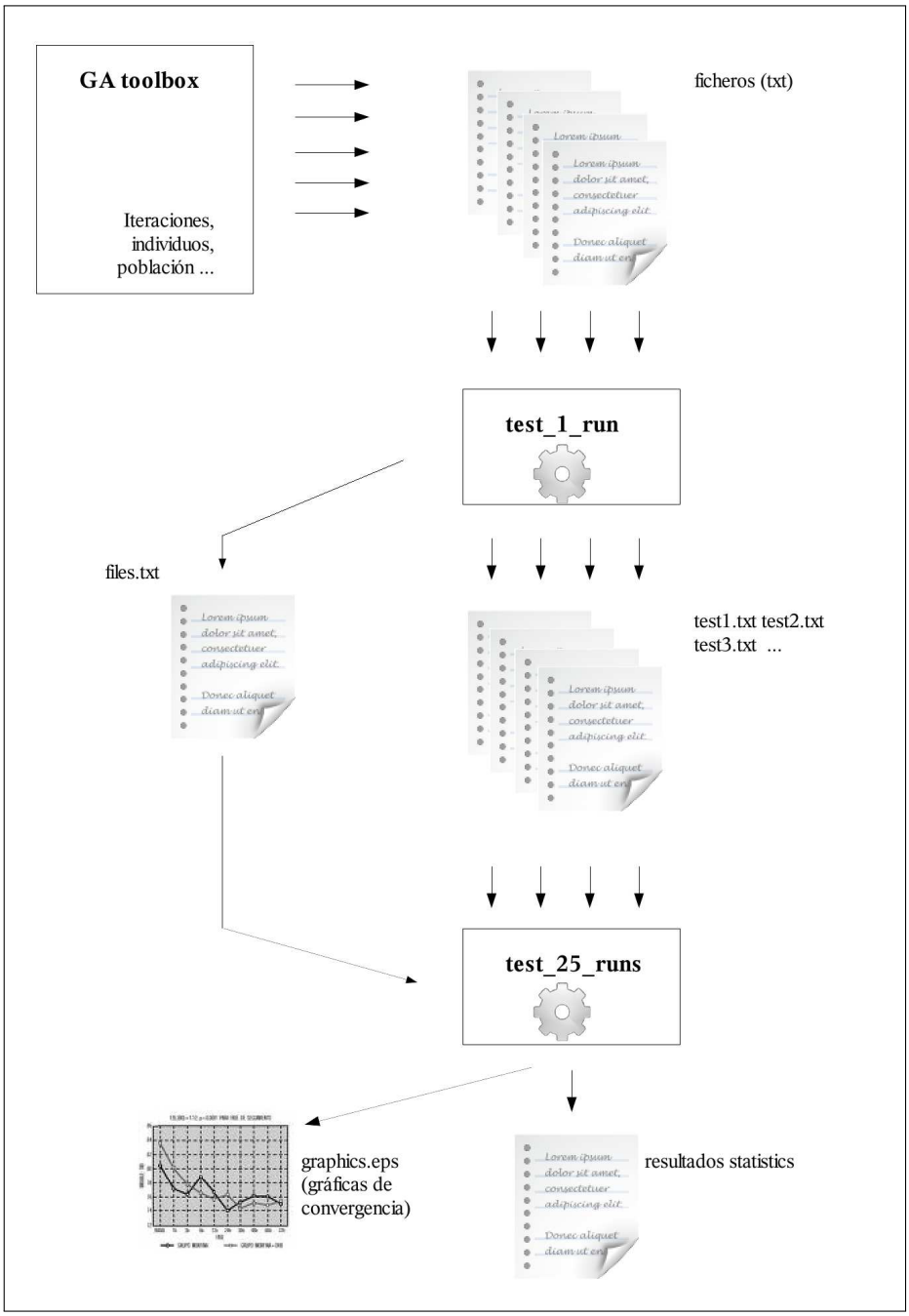

Figura 4.9: Procesamiento de datos: programa Statistics

La sinopsis para `test_1_run`, es la siguiente:

```
test_1_run number_of_FEs inputfile outputfile
```

En la ejecución de `test_1_run`, este programa, trabaja con el número de evaluaciones `number_of_FEs` y los datos de entrada `inputfile` que, tal como se indica en el apartado 4.2.4.1, son volcados a un fichero por MGA toolbox. Se recuerda que estos datos de entrada son:

- FES: máximo número de evaluaciones de la función objetivo para las cuales, el programa finaliza (solo para criterio de parada *Número de evaluaciones limitado*, ver sección 4.1) y que no es otro que el modo de finalización Max_FES
- Tamaño de la población
- Generación
- Máximo valor de salud en la población (en cada generación)
- Valor de la función(es) objetivo en cada generación

Con esto, `test_1_run` se procesa y se vuelca a otro nuevo fichero `outputfile` las siguientes estadísticas:

- El mínimo valor de error de todas las iteraciones.
- Número de evaluaciones requerido para alcanzar el nivel de precisión.
- Modo de finalización (0 Max_FES para y 1 para Ter_Err).
- Número de FES hasta la finalización (ya sea por Max_FES o Ter_Err)
- Error de la función $(f(x) - f(x^*))$ en cada iteración.
- Logaritmo del error de la función en cada iteración.

Es necesario que en cada una de las 25 ejecuciones, el nombre de estos nuevos ficheros sea distinto. En caso de que el nombre del nuevo fichero coincida con uno existente, el programa da un mensaje de error y no continua. En nuestro caso se han utilizado: `test_1.txt, test_2.txt, test_3.txt, ..., test_25.txt`. Además se crea un fichero `files.txt` (en el mismo directorio de los ejecutables) donde se guarda el nombre, según orden de creación, de cada uno de estos 25 ficheros, que servirá a `test_25_runs` como guía para indicar donde se encuentran (ruta y nombre).

Si por algún casual, se ejecutara `test_1_run`, más de 25 veces (evidentemente, con todos los nombres para `output` distintos), el programa detectaría el error, lo muestra con un mensaje y para la ejecución.

La sinopsis para `test_25_runs` es:

```
test_25_runs number_of_FEs outputfile
```

Como se acaba de comentar, a partir del fichero `files.txt` que sirve como guía, `test_25_runs` lee los 25 ficheros creados por `test_1_run` en cada una de las ejecuciones. Con estos datos realiza las siguientes estadísticas:

- Entre las 25 ejecuciones del algoritmo, el 1º (el mejor), el 7º, el 13º , el 19º y el 25º (el peor) mínimo valor de error de todas las FES (en cada ejecución), su media y su desviación estándar.
- Entre las 25 ejecuciones del algoritmo, el 1º (el mejor), el 7º, el 13º , el 19º y el 25º (el peor) número de evaluaciones requerido para alcanzar el nivel de precisión y su clasificación (ranking).
- Entre las 25 ejecuciones del algoritmo, el 1º (el mejor), el 7º, el 13º , el 19º y el 25º (el peor) valor de error de las 25 ejecuciones, después de $1e3$, $1e4$, $1e5$ FES y al final de la ejecución, su media y su desviación estándar.
- Ratio de éxito y rendimiento de éxito.
- Rendimiento medio del total de ejecuciones finalizadas (Max_FES o Ter_Err), necesario para las gráficas: $log_{10} = f(x) - f(x^*)$).
- Gráficas de convergencia

`test_25_runs` vuelca estas estadísticas en un fichero, `output`, para que puedan ser manipuladas, observadas o simplemente almacenadas.

Los datos necesarios para representar las gráficas de convergencia son almacenados en el fichero `graphics.txt`, que el software crea en el propio directorio. Para la representación el programa gnuplot es llamado por el ejecutable `test_25_runs`, que siguiendo las instrucciones del script `script.p` representa las gráficas de convergencia y las guarda en el archivo gráfico PostScript encapsulado `graphics.eps`. La siguiente figura 4.10 detalla este proceso de representación.

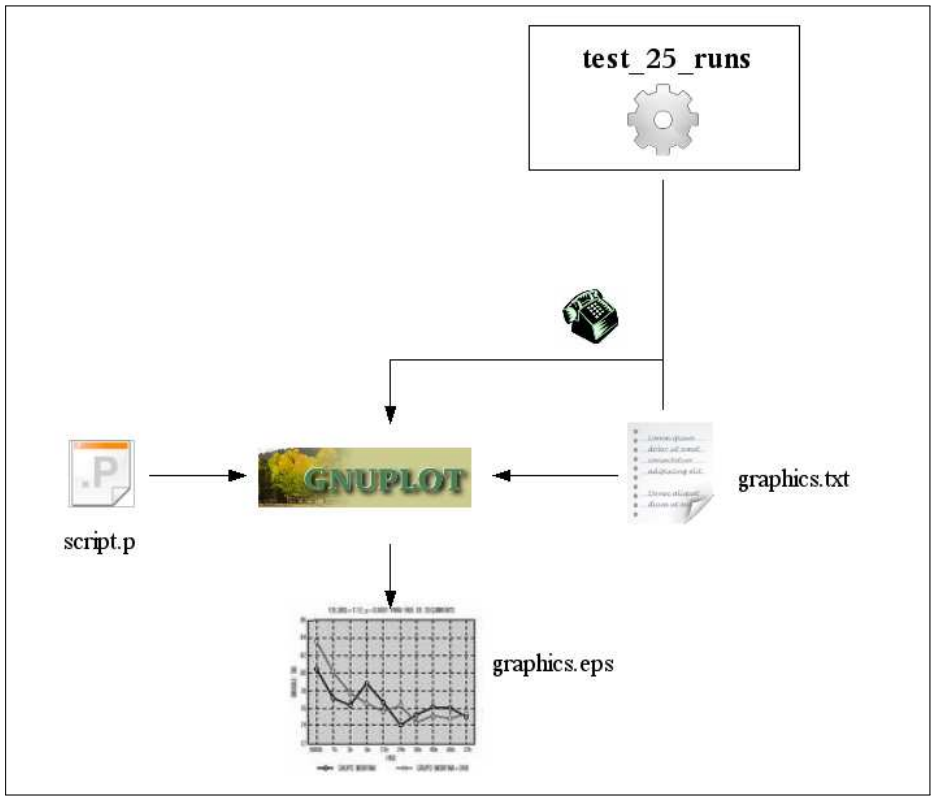

Figura 4.10: Proceso de representación: programa Statistics

Al final de la ejecución, `test_25_runs` borra los ficheros creados por `test_1_run` y el fichero `files.txt`. Si se desea recalcular las estadísticas, se debe realizar todo el proceso desde el principio: ejecutar `test_1_run` cada una de las 25 veces que se ejecuta el algoritmo genético para coleccionar datos y estadísticas de cada una de estas y al final de las 25 ejecuciones, utilizar `test_25_runs`.

4.3.5. Código programa Statistics

En este apartado, se muestra una descripción de cada uno de los ficheros que componen el programa Statistics. Los ficheros `gnuplot.cpp` y `statistics.cpp` son los únicos que tienen su correspondiente fichero de cabecera `.hpp` que contienen las definiciones de las clases. Además en el fichero `statistics.hpp` se encuentran las macros `# define f*` correspondientes a cada función.

`main_1_run.cpp` Contiene la función `main()` del programa `test_1_run`.

`main_25_runs.cpp` Contiene la función `main()` del programa `test_25_runs`.

`statistics.cpp` Contiene la implementación de la clase `Statistics`, la cual se encarga de llevar a cabo las labores de lectura de ficheros, escritura a ficheros, operaciones, manejos y cálculo de estadísticas tanto para el programa `test_1_run` como para el `test_25_runs`.

`gnuplot.cpp` Contiene la implementación de la clase `GNUplot`. Esta se utiliza para la realizar la representación de las gráficas y se encarga de llamar al programa GNUplot.

`script.p` Scrip que contiene los comandos para la representación de las gráficas (es llamado por este programa). Los comandos de este script están en el lenguaje que usa GNUplot.

La siguiente figura 4.11 muestra relación que existe entre ficheros del programa Statistics.

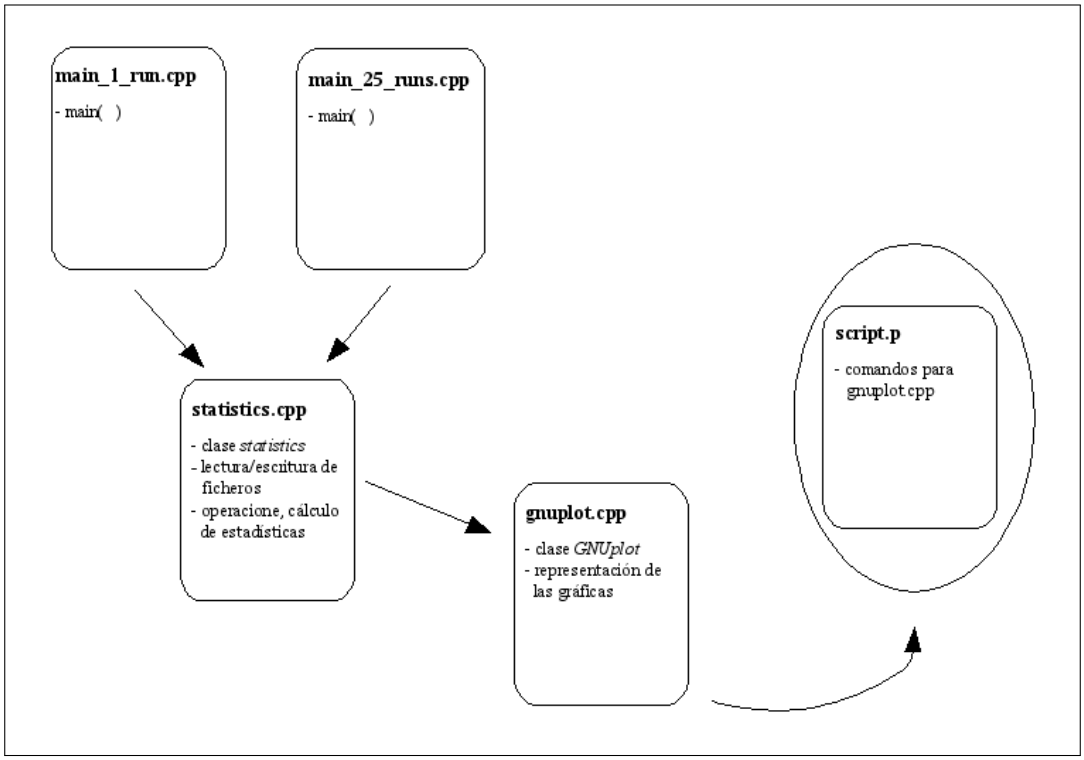

Figura 4.11: Relación entre ficheros. Programa Statistics

Entre las funciones a destacar que contiene `statistics.cpp` tendríamos:

- `void statistics_1_run(char* argv[])`: bucle principal del ejecutable `test_1_run`. Realiza cálculos estadísticos y las llamadas a las funciones necesarias para leer información de los ficheros de las ejecuciones del algoritmo genético (MGA toolbox), obtener estadísticas y volcar información al fichero `output` y `files.txt`.

- `void statistics_25_runs(char* argv[])`: bucle principal del ejecutable `test_25_runs`. Realiza las llamadas a las funciones necesarias para obtener estadísticas generales (de las 25 ejecuciones), las estadísticas para la representación gráfica y volcar información al fichero `output` y `graphics.txt`. Además lee el fichero `files.txt`.

- `void rank(double* array,int n)`: clasifica los elementos de un array de menor a mayor. Esta función no devuelve ninguna variable ni array.

- `double minimum(double* array,int n)`: calcula el mínimo elemento de un array de dimensión *n* y lo devuelve como un variable tipo *double*.

- `double mean(double* array,int n)`: calcula la media de los elementos de un array de dimensión *n* y lo devuelve como un variable tipo *double*.

- `double meangraphics(void)`: calcula los datos de coordenadas necesarios para la representación gráfica realizando una media de cada iteración del array `all_logerror_value` (una media de las 25 ejecuciones del valor del error en cada iteración).

- `double std(double* array, int n, double mean)`: calcula a partir de la media, la desviación estándar de los elementos de un array de dimensión *n* y lo devuelve como un variable tipo *double*.

- `double calculate_rates(void)`: calcula los ratios que se indican en la sección 4.3 y los carga en las variables `success_rate` y `success_performance`.

- `void setvalues(void)`: contiene las directivas condicionales #ifdef que comprueba qué macro-identificador `# define f*` del fichero cabecera `statistics.hpp` está definido para así asignar el valor *óptimo de la función* y *precisión* a sus respectivas variables.

- `void loadResultsFromFile_1_run(char *argv[])`: lee los datos de los ficheros de texto generados en cada una de las ejecuciones del algoritmo genético.
- `void loadResultsFromFile_25_runs(void)`: lee los datos de los ficheros de texto generados en cada una de las ejecuciones del ejecutable `test_1_run`, siguiendo los ficheros almacenados en `files.txt`.
- `void saveToFile_1_run(char* argv[])`: crea un fichero con nombre `output` y guarda las estadísticas calculadas en `void statistics_1_run(char* argv[])`. Además guarda en el fichero `files.txt` el nombre del fichero creado.
- `void saveToFile_25_runs(char* argv[])`: crea un fichero con nombre `output` y guarda las estadísticas calculadas en `void statistics_1_run(char* argv[])`. Además crea el fichero `graphics.txt` para la representación gráfica.

4.3.6. Software Algorithm Complexity

Para procesar el 5º criterio de evaluación del PDEC-05, Algorithm Complexity, se han seguido los pasos detallados de la documentación [SHL+05]. Así se han desarrollado los programas *computing_time_t0*, *computing_time_t1* y *computing_time_t2* que procesan los tiempos T_0, T_1 y T_2 respectivamente. El contenido de este software se detalla a continuación.

computing_time_t0 Sencillo programa que calcula el tiempo que se tarda en ejecutar lo siguiente:

```
for i=1:1000000computing\_time\_t0
  x= (double) 5.55;
  x=x + x; x=x./2; x=x*x; x=sqrt(x); x=ln(x); x=exp(x); y=x/x;
end
```

Este tiempo de ejecución de este se define como T_0. A continuación se muestra la implementación de este programa.

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <sys/times.h>
# include <iostream>

using namespace std;

int main()
{
```

```
  int i;
  double x,y;
  long inicio, fin;
  float t;

  inicio=clock();
    for (i=0;i<1000000;i++){
      x=5.55;
      x+=x;
      x=x/2.00;
      x*=x;
      x=sqrt(x);
      x=log(x);
      x=exp(x);
      y=x/x;
    }
  fin=clock();

  t=(fin-inicio);
  t=t/CLOCKS_PER_SEC;

  cout<<."El tiempo de ejecucion del proceso es de.."<<t<<" segundos"<< endl;
  t=t*1000;
  cout<<."El tiempo de ejecucion del proceso es de.."<<t<<" milisegundos"<< endl;
}
```

Como se puede ver, para calcular el tiempo de ejecución se ha recurrido a utilizar la función `clock()` que se incluye en el archivo de cabecera de la biblioteca estándar del lenguaje de programación C `time.h` (contiene funciones para manipular y formatear la fecha y hora del sistema) y que devuelve el número de pulsos de reloj desde que se inició el proceso. Los pasos seguidos son:

1. Tomar el valor del reloj antes de realizar la llamada (inicio).

2. Llamar a la rutina en cuestión.

3. Tomar nuevamente el valor del reloj (fin).

4. Calcular diferencia entre $fin-inicio$, que nos devuelve el número de pulsos de reloj total.

5. Dividir esta diferencia por `CLOCKS_PER_SEC` (constante que define el número de pulsos de reloj por segundo) obtenemos finalmente el tiempo de ejecución del proceso.

La compilación y ejecución de este programa sigue el procedimiento estándar de un programa de C++ sencillo, según el software y sistema operativo que se disponga. El nombre del ejecutable obtenido es `computing_time_t0`

computing_time_t1 Programa en C++ que obtiene el tiempo de ejecución de 200000 evaluaciones (solamente evaluaciones, no con el resto de algoritmo) de la función 3 para una determinada dimensión D. Esto nos da T_1.

El método para calcular el tiempo es el mismo que para computing_time_t0, es decir, utilizando la función `clock()` y siguiendo los mismos pasos. Las llamadas a la función 4 se realizan por medio de la siguiente función:

```
long double calc_benchmark_func(long double *x);
```

A esta función se la pasa un array x con el valor de las variables. El resto del código correspondiente a la función 4, proviene del programa que se puede descargar de `http://www3.ntu.edu.sg/home/EPNSugan/` y, cuyas funciones, se han adaptado y traspasado a computing_time_t1.

La función 3 lee datos de un fichero de datos asociado (que también se encuentra en el software descargable del PDEC-05). Este fichero se han ubicado dentro de la carpeta `input_data` en el directorio del programa.

La compilación y ejecución es igual que para computing_time_t0. El ejecutable obtenido será `computing_time_t1`.

computing_time_t2 Se trata del software MGA toolbox al que se le ha añadido un el mismo contador de tiempo utilizado en computing_time_t0 y computing_time_t1 para obtener el tiempo de ejecución completo del algoritmo para la función 3 con 200000 evaluaciones y la misma dimensión D, que corresponde a T_2.

Para ello se han implementado la función `clock()` y se ha insertado en el main, dentro del fichero `Userdefinables.cpp`, como se puede ver

```
int main(int argc, char *argv[]) {
inicio=clock();
  int ii;
  const int ciBufSize = 1024;
  char *pToken, caBuf[ciBufSize];
  FILE *fInput, *fOutput;

...

  if(globalSetup->loadPopulation)
    delete [](globalSetup->populationFileName);
  if(globalSetup->savePopulation)
    delete [](globalSetup->saveEvalSolutions);
  if(globalSetup->saveResults)
    delete [](globalSetup->saveEvalResults);
fin=clock();
```

```
t=(fin-inicio);
t=t/CLOCKS_PER_SEC;
cout<<."El tiempo de ejecucion del proceso es de.."<<t<<" segundos"<< endl;
t=t*1000;
cout<<."El tiempo de ejecucion del proceso es de.."<<t<<" milisegundos"<< endl;

  return 1;
}
```

En cuanto a la compilación de estos programas se realiza del mismo modo que para GA toolbox (consulte la documentación [Sas07]). Para la ejecución, es también igual (ver apartado 4.1.2), salvo que el ejecutable obtenido tras la compilación recibe el nombre de `MGAtbx`.

Capítulo 5

Experimentación

Una vez alcanzados los objetivos instrumentales y finalizada la adaptación del software de algoritmos genéticos MGA Toolbox, así como la programación del software Statistics para la recopilación de los datos (capítulos 3 y 4), se procederá a realizar la experimentación, objetivo principal del proyecto.

En este capítulo se va a estudiar el nuevo operador selección basado en la fórmula electoral escogida, la ley d'Hondt, por medio de la comparación con la ruleta en el AGS, según marca el objetivo principal. A continuación se definen cómo se van a realizar el experimento y las estrategias tomadas. En la última sección se muestran los resultados más relevantes. Además, en el apéndice F se recogen todos los resultados obtenidos.

5.1. Diseño del experimento

El objetivo del diseño del experimento, se corresponde con el objetivo principal del proyecto (ver sección 1.2) que es *comparar el comportamiento del algoritmo genético simple -con el operador Ruleta- en la resolución de un problema de optimización representativo, frente a una variante del algoritmo genético simple que incorpore un nuevo operador selección basado en alguna de las fórmulas electorales de divisores comunes.*

Para el algoritmo existen varios parámetros definidos como opciones de comparación (ver además apartado 5.2.2), que necesitan ser ajustados. Estos parámetros o grados de libertad serán definidos por el objetivo principal y el procedimiento de evaluación.

Atendiendo a la condición de AGS (apartado 2.1.5), impuesta en el objetivo

principal, a priori habría que definir la representación, el tamaño población, cómo realizar la evaluación, la probabilidad de cruce y la probabilidad de mutación. Para el AGS, la probabilidad de cruce es siempre igual a uno, $Pc = 1$. En lo que respecta a la representación y evaluación, en el apartado 4.2.1.2 se implementa la codificación para la representación, mientras que en el apartado 5.2.2.1 se justifica la selección de opciones para la función de salud (método de escalado: truncado y método de gestión de restricciones: no hay restricciones). Por lo tanto solamente quedan como grados de libertad relacionados con el empleo del AGS, el tamaño de población n y la probabilidad de mutación Pm. A estos grados de libertad los hemos agrupado como parámetros del algoritmo.

En el procedimiento de evaluación escogido, el PDEC-05 [SHL$^+$05], se pide definir la dimensión de la función objetivo (complejidad) d y la función a estudiar. Por otro lado, partiendo del objetivo principal, la comparación se realiza para el operador selección ruleta y el nuevo operador (basado en alguna de las fórmulas electorales de divisores comunes), para un problema de optimización. Por ello, los grados de libertad relativos al procedimiento de evaluación y el objetivo principal, serían la dimensión de la función objetivo y el operador selección, que serán englobados como parámetros del problema.

A continuación se recogen los anteriores parámetros:

- Parámetros del algoritmo
    1. Tamaño de población, n.
    2. Probabilidad de mutación Pm.
- Parámetros del problema
    3. Dimensión de la función objetivo (complejidad), d.
    4. Operador selección: operador selección ruleta R y operador selección d'Hondt D.

Partiendo de esta definición de parámetros y teniendo en cuenta que el PDEC-05 ([SHL$^+$05] y apéndice D apartado D.3.3, procedimiento de evaluación escogido, apartado 3.2.1) enuncia que el problema se ejecutará 25 veces, denominaremos **estudio** al conjunto de estas 25 ejecuciones del algoritmo bajo una determinada combinación de d, método de selección, n y Pm.

En los siguientes párrafos abordaremos primeramente la realización del diseño del experimento en sí. Esta conlleva la combinación y definición de parámetros, la determinación de los valores de estos y el desarrollo de la ejecución. Seguidamente

se aporta un ejemplo para facilitar la comprensión del diseño del experimento. Para finalizar se presentan algunas estrategias a seguir durante la experimentación.

5.1.1. Realización del diseño del experimento

Para la realización del diseño del experimento, antes de nada se deberá determinar cómo combinar los parámetros. Una vez determinada la combinación habrá que decidir los valores con los que se va a trabajar y cómo desarrollar las ejecuciones.

1. **Combinación y definición de parámetros**

   El primer paso es definir el orden de la combinación de estos parámetros o grados de libertad en los estudios, con el fin de facilitar el **análisis de los resultados**.

   Para hacer más entendible el diseño del experimento, nos ayudaremos de la tabla 5.1 de cuatro filas (uno por cada grado de libertad), que muestra las posibles combinaciones o relaciones de los grados de libertad, para cada estudio, que se pueden tomar. En ella, cada grado de libertad ocupa una fila y cada valor que puede tomar cada grado de libertad ocupa una casilla en su correspondiente fila. La relación se establece entre el valor de una casilla y uno de los valores de las casillas que quedan justo por debajo de esta. Así relacionados de arriba a abajo, una combinación se compone de 4 valores, uno por cada grado de libertad.

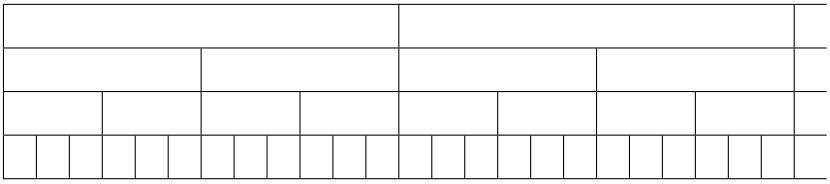

Tabla 5.1: Ejemplo tabla del diseño con 4 filas

   Para rellenar la tabla 5.1 comenzamos con los dos primeros parámetros que vienen impuestos por el propio problema. La complejidad d tomará, según el PDEC-05, los siguientes valores $d = 10$ (baja complejidad), $d = 30$ (complejidad intermedia) y $d = 50$ (alta complejidad). El operador selección, tendrá dos opciones disponibles (acorde con el objetivo principal del proyecto, sección 1.2): ruleta (R) y d'Hondt (D).

   En cuanto a la complejidad del problema, analizaremos **la complejidad baja, la media y alta, cada una por su lado, para cada uno de los métodos**

de selección propuestos. Nuestra tabla quedará con estos dos parámetros, del siguiente modo (tabla 5.2):

$d = 10$			
D		R	
$d = 30$			
D		R	
$d = 50$			
D		R	

Tabla 5.2: Tabla del diseño del experimento con los parámetros del problema

Los parámetros del algoritmo (el tamaño de población n y la probabilidad de mutación Pm) no vienen definidos a priori. Para el experimento, cada parámetro tomará una serie de valores que se define como $n = [n_1, n_2, \ldots, n_m]$ y $Pm = [Pm_1, Pm_2, \ldots, Pm_n]$. Según su naturaleza, cada tamaño de población n_i puede tomar valores pares dentro del dominio $A_i = [2, \propto] \in \mathbb{N}$. Por otro lado cada probabilidad de mutación Pm_j puede tomar valores dentro del dominio $B_i = [0, 1] \in \mathbb{R}$. La relación que se establece entre los dos parámetros es que **con cada tamaño de población n_j se analizará cada una de las n probabilidades de mutación.**

Añadiendo estos dos parámetros a la tabla 5.2 tendremos la tabla inicial 5.1 con todos los parámetros (del problema y el algoritmo).

La relación completa de los 4 parámetros quedaría por tanto del siguiente modo **cada complejidad se analizará cada uno de los métodos de selección propuesto, con cada tamaño de población y con cada probabilidad de mutación**. Como ejemplo de combinación (se recuerda que de arriba a abajo), partiendo de la tabla 5.3, tendríamos: $d = 10$, R, n_2, Pm_2. Queda definida, por tanto, la combinación de parámetros del problema.

$d = 10$														
D														
n_1				n_2				...				n_m		
Pm_1	Pm_2	...	Pm_n	Pm_1	Pm_2	...	Pm_n	Pm_1	Pm_2	...	Pm_n	Pm_1	Pm_2	...
$d = 10$														
R														
n_1				n_2				...				n_m		
Pm_1	Pm_2	...	Pm_n	Pm_1	Pm_2	...	Pm_n	Pm_1	Pm_2	...	Pm_n	Pm_1	Pm_2	...

Tabla 5.3: Tabla diseño del experimento con todos los parámetros (para $d = 10$)

2. **Determinación de valores de los parámetros del algoritmo**

   Como se ha visto en el paso anterior (combinación y definición de parámetros), el número de estudios posibles a realizar resulta infinito, dado por las combinaciones de los parámetros. No queda más remedio que limitar la experimentación a un número finito de estudios, que además de reflejar con claridad el comportamiento de cada operador (ruleta o d'Hondt) sea factible de realizar en tiempo (evitar tiempos excesivos de experimentación).

   De este paso anterior los parámetros del problema (complejidad (d) y operador selección) están ya definidos. Los parámetros del algoritmo quedarían como:

   - El tamaño de la población n puede tomar valores pares dentro del dominio $A_i = [2, \propto] \in \mathbb{N}$.
   - La probabilidad de mutación Pm se encuentra en el dominio $B_i = [0, 1] \in \mathbb{R}$.

   En el caso de la probabilidad de mutación, se ha considerado tomar tres valores representativos a priori, con el fin de acotar el experimento. La cuestión que se plantea ahora es qué valores escoger. Para barrer el dominio B_i, se ha decidido escoger una tasa de mutación baja o muy baja, otra media y por último una alta. La elección de una tasa de mutación baja o muy baja es para asegurar que probabilidades bajas habituales no sean menores que esta (ya que estarían muy próximas al $Pm = 0$).

   Consultando el Goldberg [Gol89a], en el capítulo 1 (sección *"A simulation by Hand"*) y en el capítulo 4 (sección *"De Jong and Function Optimization"*), se utilizan diferentes valores de probabilidad de mutación tales como $Pm = 0{,}001$, $Pm = 0{,}005$, $Pm = 0{,}01$, $Pm = 0{,}02$ y $Pm = 0{,}1$. Partiendo de estos valores, se podría establecer como probabilidad de

mutación muy baja, una inferior a la mínima utilizada que es $Pm = 0{,}001$ (evitar probabilidades menores habituales), con lo que se ha escogido la mitad: $Pm_a = 0{,}0005$. Para la probabilidad baja y alta, como se ve en el mencionado capítulo *"De Jong and Function Optimization"* hay dos valores son el resultado de multiplicar por 10 y por 100 el valor inicial de la probabilidad más baja ($Pm = 0{,}01$ y $Pm = 0{,}1$). En nuestro caso, para la probabilidad media y alta, se ha aplicado el mismo criterio, es decir $Pm_b = 0{,}005$ y $Pm_c = 0{,}05$. Con estos tres valores, queda cubierto el rango mencionado (ver figura 5.1), aunque se deja abierta la posibilidad de incluir nuevos valores si fuera necesario.

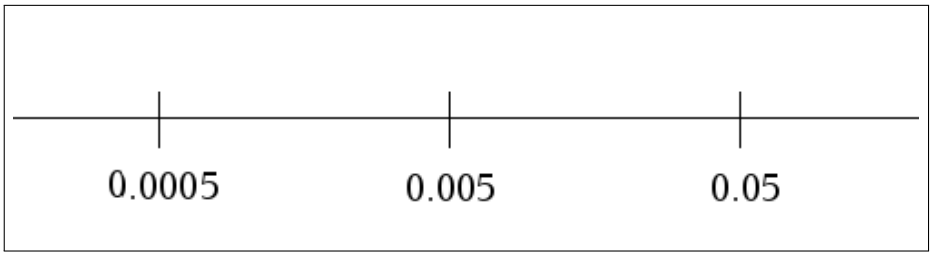

Figura 5.1: Rango de probabilidades de mutación estudiado

En lo que respecta al tamaño de población, en el apartado 5.3.2, se vieron los posibles valores del tamaño de población que se podían adoptar, según el análisis del PDEC-05. Si contamos el número de estudios a realizar, para cada una de las dos complejidades, en cada uno de los dos métodos de selección propuestos, se tomará cada uno de los dieciocho tamaños de población, para las tres probabilidades de mutación, nos salen 144 estudios. Por eso se ha decidido que de todos estos posibles valores, representados en la tabla 5.9 apartado 5.3.2, se tomarán dos (n_a y n_b), para el análisis de los resultados. El método para determinar estos dos tamaños de población se describe en el siguiente paso, desarrollo del experimento. Con ello se completa la tabla del diseño del experimento, quedando como la siguiente tabla 5.4.

Por lo tanto el **análisis de resultados** consistirá en recoger los resultados de los estudios definidos por la tabla 5.4, para comparar y obtener conclusiones de cómo se comporta el algoritmo en cada complejidad para cada uno de los operadores selección, según el tamaño de población n y la probabilidad de mutación Pm. Se analizará cada complejidad por separado y posteriormente se compararán las dos complejidades.

$d = 10$												
D						R						
$n_{(10,D)a}$			$n_{(10,D)b}$			$n_{(10,R)a}$			$n_{(10,R)b}$			
Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	
$d = 30$												
D						R						
$n_{(30,D)a}$			$n_{(30,D)b}$			$n_{(30,R)a}$			$n_{(30,R)b}$			
Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	
$d = 50$												
D						R						
$n_{(50,D)a}$			$n_{(50,D)b}$			$n_{(50,R)a}$			$n_{(50,R)b}$			
Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	

Tabla 5.4: Tabla del diseño del experimento, final

3. **Desarrollo de la ejecución**

   El último paso del diseño del experimento es el desarrollo de la ejecución que trata de definir qué operaciones hay que realizar una vez determinados los parámetros, sus valores y su relación. A continuación se detallan los pasos en el desarrollo y los estudios asociados:

   a) Seleccionar la complejidad d: alta, media o baja.

   b) Seleccionar el operador selección: d'Hondt o ruleta.

   c) Realización de estudios para determinar el tamaño de población: el objetivo es ir ejecutando el algoritmo con sucesivos tamaños de población (n), los que se crea necesario, con el fin de analizar dónde se comporta mejor, es decir, cuáles son los dos tamaños de población, n_a y n_b, que mejores resultados ofrecen en cada estudio. Esto bajo la probabilidad de mutación Pm_a. El resto de estudios no seleccionados, se descarta.

   d) Realización de estudios para el análisis del operador: con los dos tamaños de población, n_a y n_b, para los cuales se obtienen los mejores resultados con probabilidad de mutación Pm_a, se ejecuta el algoritmo con las probabilidades de mutación mencionadas (Pm_b y Pm_c).

   Una vez realizados estos pasos para las dos complejidades y operadores selección, es decir, todas las combinaciones según la tabla 5.4, se procederá a analizar los resultados.

5.1.2. Ejemplo de ejecución

A continuación se muestra un ejemplo, para la complejidad baja ($d = 10$).

Tras realizar estudios con los diversos tamaños de población con Pm_a, por un lado, para el operador selección d'Hondt (D), y por otro lado, para operador selección ruleta (R), se obtienen los dos mejores tamaños de población (descartando el resto) que se muestran en la siguiente tabla 5.5.

$d = 10$												
D						R						
$n_{(10,D)a} = 200$			$n_{(10,D)b} = 400$			$n_{(10,R)a} = 250$			$n_{(10,R)b} = 500$			
Pm_a			Pm_a			Pm_a						

Tabla 5.5: Ejemplo del diseño del experimento para Pm_a

Una vez obtenidos los tamaños de población, se realizan los estudios sobre el resto de probabilidades de mutación Pm_b y Pm_c, al igual que en el caso anterior, para cada uno de los operadores de selección por separado.

$d = 10$												
D						R						
$n_{(10,D)a} = 200$			$n_{(10,D)b} = 400$			$n_{(10,R)a} = 250$			$n_{(10,R)b} = 500$			
Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	Pm_a	Pm_b	Pm_c	

Tabla 5.6: Ejemplo del diseño del experimento completo

Ya con todos los estudios de la complejidad baja (tabla 5.6), se analizarían los resultados comparando cómo se comporta el algoritmo en esta complejidad para cada uno de los operadores selección, según el tamaño de población n y la probabilidad de mutación Pm.

Estos pasos se repetirían para la complejidad media y alta.

5.1.3. Estrategias durante la ejecución

En este apartado se muestran algunas estrategias, que se pueden tomar durante la experimentación, enfocadas a mejorar el análisis de los resultados o a disminuir el

número de estudios con el fin de aligerar la experimentación. El objetivo principal de estas estrategias es afinar el diseño del experimento con la premisa de hacer una experimentación exhaustiva y evitar tiempos excesivos en una experimentación superficial.

Probabilidad de mutación A la hora de realizar el experimento, se comenzará con la probabilidad de mutación muy baja, seguido de los estudios en la baja y por último en la alta. Pudiera ser que al observar los resultados durante la experimentación, con algún valor intermedio de Pm a los tres marcados o con un valor superior a Pm_c, se intuyera que se pudieran obtener mejores resultados. En tal caso se reforzaría la experimentación con adicionales probabilidades de mutación Pm teniendo en cuenta el estudio de De Jong [Gol89a].

Complejidad del algoritmo En lo relacionado a la complejidad del algoritmo, hay definidas tres complejidades: $d = 10$ (baja complejidad), $d = 30$ (complejidad intermedia) y $d = 50$ (alta complejidad). La estrategia consistirá en realizar primeramente los estudios de la baja y la alta complejidad. Si hay superioridad de uno de los métodos de selección en los estudios de complejidad $d = 10$ y en los de complejidad $d = 50$, podemos pensar que también se va a dar esta superioridad en la complejidad $d = 30$. Así podríamos evitar los estudios en la complejidad intermedia y analizar así mejor los resultados obtenidos en la complejidad baja y alta.

5.2. Preparación de la ejecución

En esta sección se va a mostrar cómo se ha llevado a cabo el proceso de preparación de escenarios, ficheros... para acometer el proceso de experimentación, de un modo ordenado y eficaz.

5.2.1. Denominación de escenarios

Es importante no confundir "escenario" con "estudio", definido en la sección 5.1. El estudio se refiere a las ejecuciones en sí y el escenario al entorno en el que se realizan estas ejecuciones. Así denominaremos estudio, al conjunto de las 25 ejecuciones del algoritmo bajo un determinado escenario.

Para guardar un orden e identificar rápidamente las características de un escenario, basándonos en los grados de libertad enumerados en el siguiente apartado (5.2.2) y acorde con ellos, se ha seguido la siguiente nomenclatura:

- Los dos primeros dígitos muestran la dimensión de la función objetivo (función test). Según el PDEC-05 ([SHL$^+$05] y apéndice D apartado D.3.3), puede tomar una de las siguientes 3 dimensiones: $D = 10$, $D = 30$ y $D = 50$.
- El carácter que se encuentra a continuación se corresponde a el operador selección escogido, que será R para *ruleta* y D para *d'Hondt*.
- El siguiente grupo de dígitos indica el tamaño de la población, $n =$ número de individuos.
- El último grupo corresponde con la probabilidad de mutación Pm. Se representa su parte decimal, debido a que $Pm \leq 1$. Así por ejemplo, una $Pm = 0,003$, vendrá representada en la nomenclatura como 003. El caso de $Pm = 1$ no se contempla.

La siguiente figura (5.2) muestra un ejemplo:

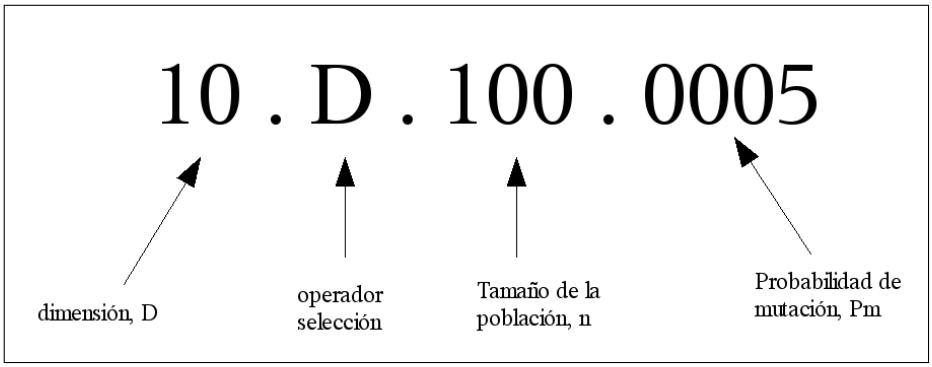

Figura 5.2: Ejemplo sistema denominación de escenarios

Grupos de escenarios Denominaremos grupo de escenarios a aquel conjunto de escenarios que comparten uno o varios grados de libertad. Esta definición nos será muy útil para manejar las ejecuciones y los resultados de la experimentación.

Así, por ejemplo, el grupo de escenarios 50.D.**.****, son todos los escenarios con complejidad $d = 50$ y operador selección d'Hondt.

5.2.2. Configuración del software para la experimentación

Para la configuración de escenarios, contamos con el fichero de entrada para definir los parámetros y opciones del software de genéticos MGA toolbox. Este

fichero se mostró en la apartado 4.1.2, solamente que en este caso, debido a las modificaciones del capítulo 4, se disponen de algunas opciones más.

Primeramente, en el software es necesario seleccionar la función objetivo escogida, que como se ha visto en el apartado 3.2.2, es la función 1, *función Esfera desplazada (De Jong 1 Desplazada)*, cuyas propiedades se pueden consultar en el apéndice D. Esto se ha de realizar antes de compilar el programa **MGA toolbox**. Según se indica en el apartado 4.2.3, dentro del fichero `userDefinables.cpp`, se llama a la función objetivo por medio de la función:

```
double suganthan_benchmark_func(double *x, int noOfDecisionVariables, int function);
```

Aquí, tenemos que escribir el número de la función, en el argumento entero `int function`, tal como se muestra a continuación:

```
...
/* Objective function and constraints go here */
void globalEvaluate(double *x, double *objArray,
double *constraintViolation, double *penalty, int *noOfViolations)
{
   int ii;
   FILE *outEvals;
   FILE *outResults;

   for(ii = 0; ii < globalSetup->finalNoOfObjectives; ii++)
     objArray[ii] = 0.0;

   if(globalSetup->gaType == SGA) {

     *objArray = myobjfunc.suganthan_benchmark_func(x,
globalSetup->noOfDecisionVariables,1);

   }
...
```

Además, en el programa **Statistics**, es necesario en el archivo de cabecera `statistics.hpp` "descomentar" la línea `# define f1` correspondiente a la función 1, como bien se comenta en el apartado 4.3.3.

En lo que respecta a la selección de opciones (parámetros) del software MGA toolbox, para el AGS y acordes con los objetivos de este proyecto, de entre todas las opciones que se encuentran más adelante, se distinguen tres grupos:

- Opciones fijas. Son opciones que son definidas para configurar al AGS.
- Opciones de comparación. Se trata de los "grados de libertad" del problema.

- Selección. Será operador ruleta para escenarios **.**.**.R y operador selección d'Hondt para escenarios **.**.**.D.
- Número de variables de decisión (dimensión). Será $d = 10$ para escenarios **.**.10.* y $d = 50$ para escenarios **.**.50.*.
- Tamaño de población.
- Probabilidad de mutación.

Por lo tanto, para configurar un escenario, se deben escoger los operadores y valores de las opciones de comparación del diseño del experimento (sección 5.1). El resto vienen impuestas.

5.2.2.1. Función de salud: selección de opciones

Como ya se comentó en el apartado 2 la función de salud da una medida de la aptitud del individuo para sobrevivir en su entorno y debe estar definida de tal forma que los individuos que representen mejores soluciones tengan valores más altos de salud. Por lo tanto, hay que configurar una función de salud que según las características tanto para el operador selección ruleta como para el d'Hondt, se comporte del siguiente modo:

1. No devuelva valores negativos
2. No sea discriminatoria
3. Pueda trabajar tanto con problemas de maximización como minimización
4. Gestione violación de restricciones

MGA toolbox nos permite actuar sobre diferentes parámetros para conseguir este comportamiento. Los parámetros son:

- Método de escalado: gestiona la función para que no devuelva valores negativos y ajusta el grado de discriminación.
- Método de gestión de restricciones

Por otro lado, MGA toolbox, distingue que tipo de problema se trata (maximización/minimización). La gestión que realiza en objetivos de minimización antes del escalado es hacer negativo el valor de la función objetivo.

Método de escalado MGA toolbox ofrece los siguientes métodos de escalado:

- Ranking lineal
- Truncado (escalado Sigma)
- Traslación

En el caso del ranking (veo que individuos forman mi población y los ordeno del mejor al peor), no se da importancia cuanto mejor sean los individuos. Por lo tanto no es nada discriminatorio y basándonos en la naturaleza del operador selección d'Hondt, está completamente desaconsejado su uso para este.

El método de la traslación es un escalado donde para: $f^* = a \cdot f + b$, $a = 0$ y para este caso particular $b = -(\text{peor valor de salud})$. Esto conlleva a que el individuo con el peor valor de salud reciba una salud escalada $f^* = 0$ y por lo tanto no pueda participar ni el operador ruleta ni en el d'Hondt. Este fenómeno no conviene ya que si tuviésemos una población muy buena al utilizar el escalado simple, perderíamos parte de la población (los individuos no "tan buenos") y sus características (que tal vez, junto a los otros podrían obtener individuos mejores). Como ejemplo imaginemos que tenemos una población con 4 individuos cuya salud sería: 100, 100, 100 y 99 (perderíamos el 99 y sus buenas características: lo que implica que los otros se cruzan consigo mismo y no mejoran).

Además, después de estudiar el código, tras un bug que ocurría para el método de selección de ruleta, se observó que para poder hacer el escalado por defecto (traslación) no se debe seleccionar el método de gestión de restricciones por torneo. A continuación se muestra el código añadido para cuando se dé la combinación de gestión de restricciones por torneo y selección por ruleta o d'Hondt, que indica el error y para el programa.

```
//The original code doesn't work correctly with the roulttewheel and D_HondtLaw
//when the Tournament constraint Method are choose.
//Therefore, this code are implemented
  if (globalSetup->constraintMethod == Tournament) {
    if(globalSetup->selectionType == RouletteWheel||
                        globalSetup->selectionType == D_HondtLaw){
    fclose(fInput);
    printf(" Error!");
    printf(Çonstraint Method for selection schemes RouletteWheel or D_HondLaw\n");
    printf("shouldn't be Tournament\n");
    printf("Please, set another Constraint Method\n");

    exit(1);
    }
  }
```

Por estos dos motivos tampoco nos conviene la opción de traslación

El truncado sin embargo ofrece un escalado de la función de salud, sin que por ello uno de los individuos siempre se vea discriminado a 0, funciona bien tanto para la ruleta como para el d'Hondt y esta implementado (se ofrece como opción) en el MGA toolbox, por lo que será la opción a elegir.

Método de gestión de restricciones MGA toolbox ofrece los siguientes métodos de escalado (sección 4.1):

- Penalización lineal
- Penalización cuadrática
- Torneo

Dado que para la función escogida, función 1 (apartado 3.2.2), no se han definido funciones frontera, escoger un método de gestión de restricciones carece de sentido.

5.2.2.2. Selección de opciones MGA toolbox

A continuación se enumeran todos los parámetros que toma el software MGA toolbox y sus respectivas opciones/valores escogidos, para cumplir con el objetivo principal del proyecto (experimentación en el AGS), el diseño del experimento (sección 5.1) y las opciones vistas en el apartado anterior 5.2.2.1:

Tipo de algoritmo genético (SGA o NSGA) Al tratarse del AGS, se escoge SGA.

Número de variables de decisión (Dimensión) Viene condicionado por el procedimiento de evaluación PDEC-05 a las dimensiones $d = 10$, $d = 30$ y $d = 50$.

Para cada variable de decisión definir: tipo de variable de decisión (double o int), límite inferior, límite superior y precisión (solo para el caso de cruce y mutación binario) Las función Esfera escogida $\in \mathbb{R}$, por lo tanto el tipo de variable a utilizar es *double*. Los límites inferior y superior vienen marcados en el PDEC-05. Por último dado que vamos a seleccionar la opción de cruce y mutación binario, debemos determinar la precisión, que como en el software que suministra el PDEC-05 maneja cifras con 4 decimales de precisión, esta será la precisión a tomar.

Objetivos: números de objetivos y qué tipo de optimización (maximización/minimización) Al tratarse de una función de minimización, Min es el tipo de objetivo elegido.

Restricciones: número de restricciones y qué peso de penalización se asigna No hay restricciones (0).

Tamaño de población En el PDEC-05 no se habla o delimita el tamaño de población. En el diseño del experimento se concreta como proceder para seleccionar el tamaño de población.

Número máximo de generaciones Número máximo de generaciones en cada ejecución. Puede servir como criterio de parada junto al escogido (ver más adelante el punto "Criterio de finalización").

Proporción de reemplazo Será total: 1(AGS).

Anidamiento (para mantener soluciones múltiples) AGS: No anidamiento (NoNiching).

Selección Se usarán dos operadores selección:

- Operador ruleta (Roulettewheel)
- Operador d'Hondt (D'Hondtlaw). Además escogeremos la barrera electoral (threshold, ver operador d'Hondt en el apartado 4.2.1.5) como 0,03 (consultar apartado 2.2.2.2, barrera electoral utilizada en España para la ley d'Hondt).

Cruce Según AGS, será el cruce simple (binario):BinaryOnePoint, con probabilidad de cruce 1.

Mutación Según AGS, mutación binaria (Binary MGABin). En el diseño del experimento se muestra la selección de la probabilidad de mutación.

Método de escalado Truncado (Sigmascaling).

Método de gestión de restricciones NoConstraints. La función propuesta no se acompaña de ninguna función frontera (restricción).

Método de búsqueda local No habrá método de búsqueda local (NoLocalSearch).

Criterio de finalización Aunque en el PDEC-05 se habla de criterios de finalización el número máximo de evaluaciones (Max_FES) o el error mínimo (Ter_Err), en los criterios de finalización ofrecidos por MGA toolbox, no se encuentra el del error mínimo. Por lo tanto, el criterio de finalización será el número máximo de evaluaciones (NoOfEvaluations). En lo que respecta al número de ventana generacional, al no ser un criterio que se basa en cambio, no es relevante el valor asignado (ver sección 4.1). Por esto se le asigna 1.

Población inicial No se carga (0). Se genera una población aleatoriamente.

Salvar las evaluaciones en un fichero No se salva las evaluaciones en un fichero (0).

Salvar los resultados devueltos por MGA toolbox en un fichero Se salvan estos resultados (para luego utilizarlos en el proceso de evaluación) en un archivo.

Finalmente el fichero de entrada quedará del siguiente modo:

```
# Input file for the MGAToolbox
# Author: Federico Egido
# Date: March, 2009
#

#
# GA type: SGA or NSGA
#
SGA

#
# Number of decision variables
#
10

#
# Decision variable type can be double or int
# For each decision variable, enter:
#  decision variable type, Lower bound, Upper bound
#
# (In case of binaryCrossover mode GABin and/or binaryMutation MGABin, enter:
#  decision variable type, Lower bound, Upper bound, Number of decimals)

double 0 100 4
double 0 100 4
double 0 100 4
double 0 100 4
double 0 100 4
```

```
double 0 100 4
double 0 100 4
double 0 100 4
double 0 100 4
double 0 100 4

#
# Objectives:
# Number of objectives
# For each objective enter the optimization type: Max or Min
#
1
Min

#
# Constraints:
# Number of constraints
# For each constraint enter a penalty weight
#
0

#
# General parameters: If these parameters are not entered default
#                     values will be chosen. However you must enter
#                     "default" in the place of the parameter.
# [population size]
# [maximum generations]
# [replace proportion]
#
40
100000
1

#
# Niching (for maintaining multiple solutions)
# To use default setting type "default"
#  Usage: Niching type, [parameter(s)...]
#  Valid Niching types and optional parameters are:
# NoNiching
# Sharing [niching radius] [scaling factor]
# RTS [Window size]
# DeterministicCrowding
#
#  When using NSGA, it must be NoNiching (OFF).
#
NoNiching

#
# Selection
#  Usage: Selection type, [parameter(s)...]
#  To use the default setting type "default"
#
#  Valid selection types and optional parameters are:
# RouletteWheel
# D_HondtLaw [threshold]
# SUS
# TournamentWOR [tournament size]
# TournamentWR [tournament size]
# Truncation [# copies]
#
#  When using NSGA, it can be neither SUS nor RouletteWheel.
#
```

```
D_HondtLaw

#
# Crossover
#  Crossover probability
#  To use the default setting type "default"
#
#  Usage: Crossover type, [parameter(s)...]
#  To use the default crossover method type "default"
#  Valid crossover types and optional parameters are
# OnePoint
# TwoPoint
# BinaryOnePoint
# BinaryTwoPoint
# Uniform [genewise swap probability]
# SBX [genewise swap probability][order of the polynomial]
#
1
BinaryOnePoint

#
# Mutation
#  Mutation probability
#  To use the default setting type "default"
#
#  Usage: Mutation type, [parameter(s)...]
#  Valid mutation types and the optional parameters are:
# Selective
#       Binary [MGABin, MCodBin]
# Polynomial [order of the polynomial]
# Genewise [sigma for gene #1][sigma for gene #2]...[sigma for gene #ell]
#
0.0005
Binary MGABin

#
# Scaling method
#  To use the default setting type "default"
#
#  Usage: Scaling method, [parameter(s)...]
#  Valid scaling methods and optional parameters are:
# NoScaling
# Ranking
# SigmaScaling [scaling parameter]
#
SigmaScaling 2

#
# Constraint-handling method
# To use the default setting type "default"
#
# Usage: Constraint handling method, [parameters(s)...]
# Valid constraint handling methods and optional parameters are
# NoConstraints
# Tournament
# Penalty [Linear|Quadratic]
#
NoConstraints

#
# Local search method
# To use the default setting type "default"
```

```
#
# Usage: localSearchMethod, [maxLocalTolerance], [maxLocalEvaluations],
# [initialLocalPenaltyParameter], [localUpdateParameter],
# [lamarckianProbability], [localSearchProbability]
#
# Valid local search methods are: NoLocalSearch and SimplexSearch
#
# For example, SimplexSearch 0.001000 20 0.500000 2.000000 0.000000 0.000000
NoLocalSearch

#
# Stopping criteria
# To use the default setting type "default"
#
# Number of stopping criterias
#
# If the number is greater than zero
#    Number of generation window
#    Stopping criterion, Criterion parameter
#
# Valid stopping criterias and the associated parameters are
# NoOfEvaluations, Maximum number of function evaluations
# FitnessVariance, Minimum fitness variance
# AverageFitness, Maximum value
# AverageObjective, Max/Min value
# ChangeInBestFitness, Minimum change
# ChangeInAvgFitness, Minimum change
# ChangeInFitnessVar, Minimum change
# ChangeInBestObjective, Minimum change
# ChangeInAvgObjective, Minimum change
# NoOfFronts (NSGA only), Minimum number
# NoOfGuysInFirstFront (NSGA only), Minimum number
# ChangeInNoOfFronts (NSGA only), Minimum change
# BestFitness (SGA with NoNiching only), Maximum value
#
1
1
NoOfEvaluations 100000

#
# Load the initial population from a file or not
# To use the default setting type "default"
#
# Usage: Load population (0|1)
#
# For example, if you want random initialization type 0
# On the other and if you want to load the initial population from a
# file, type
#  1 <population file name> [0|1]
#
# Valid options for "Load population" are 0/1
# If you type "1" you must specify the name of the file to load the
# population from. The second optional parameter which indicates
# whether to evaluate the individuals of the loaded population or not.
0

# Save the evaluated individuals to a file
#
# To use default setting type "default".
#
# Here by default all evaluated individuals are stored and you will be
# asked for a file name later when you run the executable.
```

```
#
# Usage: Save population (0|1)
# For example, if you don't want to save the evaluated solutions type 0
# On the other and if you want to save the evaluated solutions
#  1 <save file name>
#
# Note that the evaluated solutions will be appended to the file.
#
# Valid options for "Save population" are 0/1
# If you type "1" you must specify the name of the file to save the
# population to.
0

# Save the results of the runned program to a file (also called
# statistics, which are displayed when the program runs)
# To use default setting type "default".
#
# Here by default all evaluated individuals are stored and you will be
# asked for a file name later when you run the executable.
#
# Usage: Save results (0|1)
# For example, if you don't want to save the evaluated solutions type 0
# On the other and if you want to save the evaluated solutions
#  1 <save file name>
#
# Note that the results will be appended to the file.
#
# Valid options for "Save results" are 0/1
# If you type "1" you must specify the name of the file to save the
# results to.
1 resultados/resultados.txt

#END
```

5.2.3. Creación de ejecutables

En la preparación de la ejecución también se crearán los ejecutables de los programas a utilizar, según las referencias que se indican a continuación:

- *Algorithm Complexity*: `computing_time_0`, `computing_time_1` y `computing_time_2`, ver apartado 4.3.6.
- *MGA toolbox*: `MGAtbx`, ver sección 4.2.
- *Statistics*: `test_1_run` y `test_25_runs`, ver apartado 4.3.3.

Además se necesitará tener instalado GNUplot (consultar apartado 4.3.2).

5.2.4. Estructura de directorios

Con el objeto de mejorar la eficacia a la hora de realizar la ejecución, se estudió el modo de organizar, dependiendo de los escenarios y las variables, los directorios en donde se guardarían los ficheros, ejecutables y resultados. La estructura

resultante es la que se puede observar en la figura 5.3, en la que primeramente se engloban los estudios según la probabilidad de mutación y cuya organización está basada en la denominación de escenarios (consultar la figura 5.2). Estos constan de varias carpetas que recogen los escenarios pertenecientes a una determinada complejidad d y un determinado operador selección (d'Hondt o ruleta) bajo un tamaño de población variable (**) y con esa probabilidad de mutación.

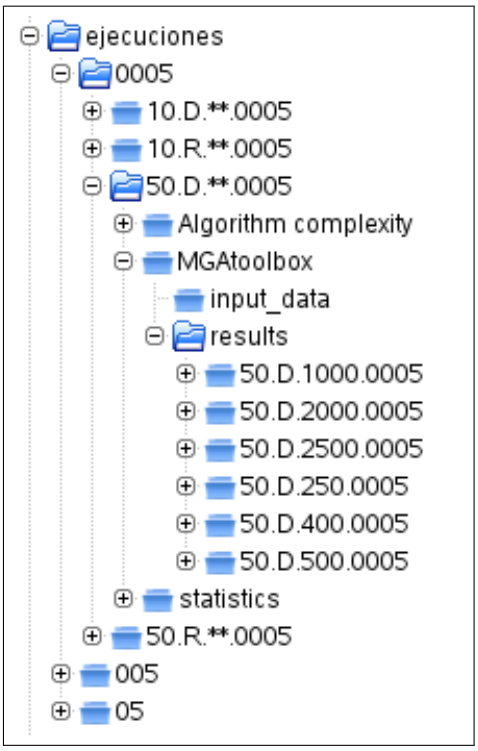

Figura 5.3: Estructura de directorios llevada

Así por ejemplo, en el directorio 50.D.**.0005 se engloban estudios para varios tamaños de población (opción de comparación a estudiar), bajo una dimensión $d = 50$, con la ley d'Hondt como operador selección (operador d'Hondt), para una $Pm = 0{,}0005$.

En cada directorio hay 3 subdirectorios que son los siguientes:

- *Algorithm Complexity*: contiene el ejecutable del software para medir el computing_time_2 (complejidad del algoritmo) y los ficheros de entrada.
- *MGA toolbox*: contiene el ejecutable del programa de algoritmos genéticos

escogido y modificado, MGAtbx, los ficheros de entrada y los resultados de las ejecuciones.

- *Statistics*: contiene los ejecutables del programa de análisis de resultados (según PDEC-05), Statistics, los scripts para la representación y los resultados que devuelven estos ejecutables.

5.3. Ejecución

5.3.1. Operaciones y pasos realizados

En este apartado se explican las operaciones realizadas durante la ejecución, partiendo del diseño del experimento, sección 5.1, siguiendo las instrucciones del PDEC-05 (apéndice D).

Anteriormente a todas las operaciones, como ya se ha comentado en la sección 5.2, se realizan los *trabajos previos* a la experimentación. Los programas se ejecutarán según se ha explicado en la sección 4.2 para MGA toolbox, sección 4.3 para Statistics y apartado 4.3.6 para Algorithm Complexity.

Cálculo de $T0$ y $T1$ Primeramente, antes de empezar con la ejecución de los escenarios (MGA toolbox), se ejecutan los programas que calculan $T0$ (computing_time_t0) y $T1$ (computing_time_t0). Los resultados obtenidos son los siguientes (tabla 5.7):

D	*T0*	*T1*
10	0,18	1,17
30		11,72
50		30,29

Tabla 5.7: T0 y T1. Algorithm Complexity

Estudios para determinar el tamaño de población Para la probabilidad de mutación Pm_a, tras seleccionar la complejidad d, se decide qué operador selección (d'Hondt o ruleta) se estudiará primero. Entonces escogemos un tamaño de población que pensemos que puede ser bueno y para ese escenario (por ejemplo el 10.D.250.0005):

1. Se ejecuta 25 veces MGA toolbox (1 estudio).
2. Ejecución de Statistics una vez finalizado el estudio, para obtener los resultados según PDEC-05.
3. Cálculo de $T2$-Algorithm complexity. Se ejecuta 5 veces el programa computing_time_t2.
4. Se recogen los resultados de Statistics y Algorithm complexity en un Excel para su análisis.
5. Analizar si se han estudiado suficiente tamaños de población para determinar cuáles son los dos tamaños de población, que mejores resultados obtienen o por el contrario es necesario estudiar otro tamaño de población distinto y repetir estos 5 pasos.

Una vez finalizado, se tienen ya los dos mejores tamaños de población, n_a y n_b, bajo la probabilidad de mutación Pm_a. El resto de estudios no seleccionados, se descarta. Se repite el proceso con el otro operador selección no estudiado.

Estudios para el análisis del operador Con los dos tamaños de población, n_a y n_b (mejores resultados con probabilidad de mutación Pm_a), se decide al igual que antes, qué operador selección (d'Hondt o ruleta) se estudiará primero y qué tamaño de población. Realizaremos para la probabilidad de mutación Pm_b y para Pm_c las siguientes operaciones en cada escenario (por ejemplo 10.R.250.05):

1. Se ejecuta 25 veces MGA toolbox (1 estudio).
2. Ejecución de Statistics una vez finalizado el estudio, para obtener los resultados según PDEC-05.
3. Se recogen los resultados de Statistics y Algorithm complexity en un Excel para su posterior análisis.
4. Cálculo de $T2$-Algorithm complexity. Se ejecuta 5 veces el programa computing_time_t2.
5. Se repiten estos 5 pasos con el otro tamaño de población.

Finalmente se repite el proceso con el otro operador selección no estudiado.

Una vez finalizados los estudios para determinar el tamaño de población y para el análisis del operador, con la primera complejidad, se procederá a dar los mismos pasos con el resto de complejidades d.

La figura 5.4 representa el flujograma del proceso general.

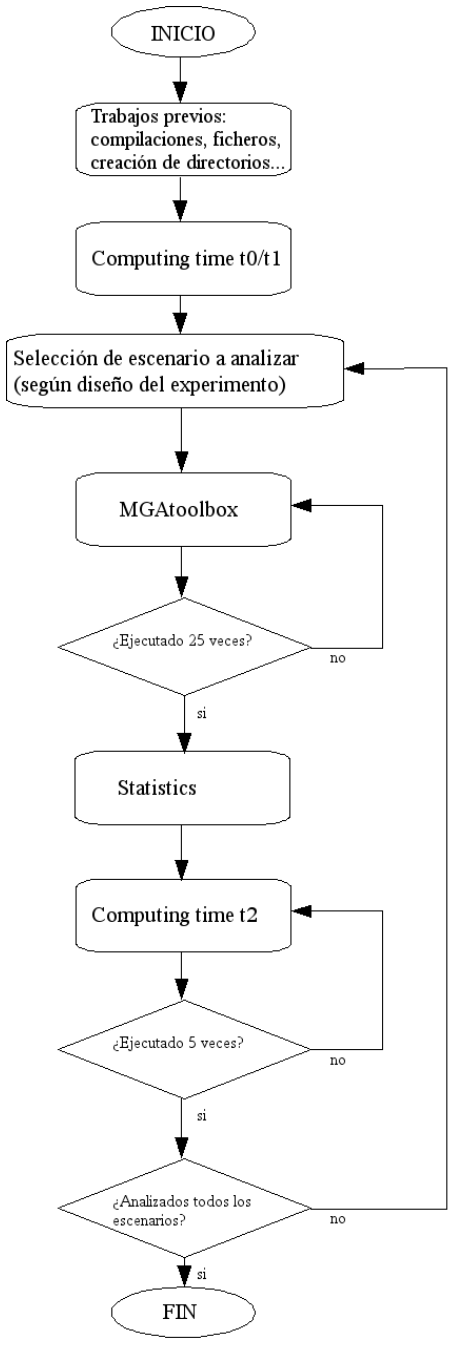

Figura 5.4: Flujograma del proceso (modificar según el 6.1)

5.3.2. Ajustes y cambios en la ejecución

En este apartado se explican los diferentes problemas que surgieron en la ejecución y como se han resuelto.

Problema tamaño de población fijo Un problema que surgió en la experimentación, con el tamaño de población, es que cuando según el PDEC-05 pide guardar el valor de error de la función para 1e3 FES, este tamaño no puede ser cualquiera, esto es debido a que lo que se busca es que coincida con algún valor exacto de la generación, por ejemplo para 250, la cuarta generación coincide con 1e3 (4*250=1000). Básicamente, en términos aritméticos, los tamaños de población que cumplen esta condición son los divisores o factores propios de 1000.

El motivo de hacer que coincidan exactamente las 1e3 FES (evaluación la función de salud) con alguna generación es que si tomamos el valor del individuo que hace los 1e3 FES, con un tamaño de población que no cumpla esta condición, factor propio de 1000 (por ejemplo 300), sale cualquier resultado o solución al azar, ya sea buena o mala (poco o nada representativo) ya que puede ser que los mejores individuos estén entre los primeros 150 o viceversa (cuando se alcancen las 1e3).

Los factores propios se recogen en la tabla 5.8. Estos representan el conjunto de tamaños de población a escoger para el estudio sobre el tamaño de población (ver apartado 5.3.1). Evidentemente no todos van a ser analizados y se irá viendo para qué tamaños de población de los siguientes se obtienen mejores resultados.

2	4	5	8	10	20	40	50	100	125	200	250	500	1000

Tabla 5.8: Factores propios de 1000

Como se ve, aparecen saltos significativos o un rango un poco limitado por arriba. Con la intención de completar estos saltos y ampliar el rango, se han añadido a los anteriores tamaños de población, n=400, n=2000, n=2500 y n=5000, contando con el inconveniente de que no son evaluables para 1e3 FES (número de evaluaciones de la función objetivo). La tabla 5.9 muestra el rango completo de tamaños de población.

2	4	5	8	10	20	40	50	100	125	200	250	400	500	1000	2000	2500	5000

Tabla 5.9: Tamaños de población elegibles

Accuracy level En relación a los criterios de evaluación del PDEC-05 (sección 4.3 y apéndice D) existen dos, Max_FES y los ratios *Ratio de éxito* y *Rdto. de éxito*, que se basan en alcanzar un determinado nivel de precisión por parte del algoritmo. Ninguna de las opciones del algoritmo genético utilizadas para este proyecto, como se verá en el siguiente apartado 5.4, alcanza la precisión comentada en la sección 4.3 (se recuerda que el nivel de precisión para de la función escogida f_1, es de -450+1e-6). Con el fin de aprovechar estos criterios de evaluación para el análisis de resultados, se decidió variar estos niveles de precisión a otros menos restrictivos. Estos niveles se muestran ya dentro de la sección de resultados 5.4.

5.4. Análisis y resultados

En esta sección se incluyen los resultados de la aplicación de la test suite PDEC-05 (sección 4.3 y apéndice D) al AGS con operador selección ruleta y con d'Hondt, tal como exige el objetivo principal del proyecto (sección 1.2), para la función escogida que es la función Esfera desplazada (apartado 3.2.2). Como complemento, en el apéndice F se recogen todos los resultados obtenidos.

Todos los experimentos han sido realizados con el software MGA toolbox (sección 4) y ejecutados en el mismo ordenador-AMD Turion(tm) 64 X2 Mobile Technology TL-58, 2,8 GB RAM en Linux 2.6.22.5-31-default x86_64, openSUSE 10.3 (x86_64), KDE: 3.5.7 release 72".

Algorithm Complexity A continuación, en la siguiente tabla 5.10, se muestran los datos de los tiempos $T0$ y $T1$, necesarios para el cálculo de la complejidad del algoritmo, obtenidos según se describió en el apartado 4.3.6.

D	T0	T1
10	0,18	1,77
30		11,72
50		30,29

Tabla 5.10: Computing_time_0 y Computing_time_1

5.4.1. Resultados baja complejidad ($d = 10$)

5.4.1.1. Determinación del tamaño de población

Para los grupos de escenarios 10.D.**.0005 y 10.R.**.0005, se han realizado estudios, con los siguientes tamaños de población: 50, 100, 200, 250, 400, 500, 1000 y 2000. En la siguiente tabla (5.11) se recogen los datos de error del estudio (mínimo error[1] y error medio) después de 1e5 evaluaciones (FES), al final de la ejecución, para cada tamaño de población n en los dos casos (operadores selección). Además, las gráficas 5.5 y 5.6 muestran la relación entre el error y el tamaño de población.

d'Hondt								
Tamaño de población	50	100	200	250	400	500	1000	2000
1st (Best)	2,43	0,44	0,0029	0,0008	0,023	0,08	0,62	3,44
Mean	151,45	62,79	3,19	3,84	1,37	0,6	3,61	11,46
Ruleta								
Tamaño de población	50	100	200	250	400	500	1000	2000
1st (Best)	19,72	7,3	14,7	8,41	6,26	7,85	9,22	10,27
Mean	368,59	186,54	56,97	47,17	26,93	24,04	22,82	37,49

Tabla 5.11: Mínimo error y error medio, tras finalizar, en los estudios de 10.D.**.0005 y 10.R.**.0005

A la vista de estos resultados, los mejores tamaños de población para $d = 10$ podrían ser en el d'Hondt $n = 200$ y $n = 250$, ya que son los que son los que menor mínimo error obtienen, aunque estos valores para $n = 400$ y $n = 500$, también son buenos y además de tienen una media de error menor. Para la ruleta $n = 400$ y $n = 500$ son los tamaños de población que menor mínimo error tienen y con valores de error medio de lo mejor. Puntualmente $n = 100$ alcanza un muy buen valor del mínimo error, pero en cambio el error medio es muy alto.

[1] El mínimo error se trata, de las 25 ejecuciones del estudio, del primer error de la función: el mejor (ver apartado 3.2.1)

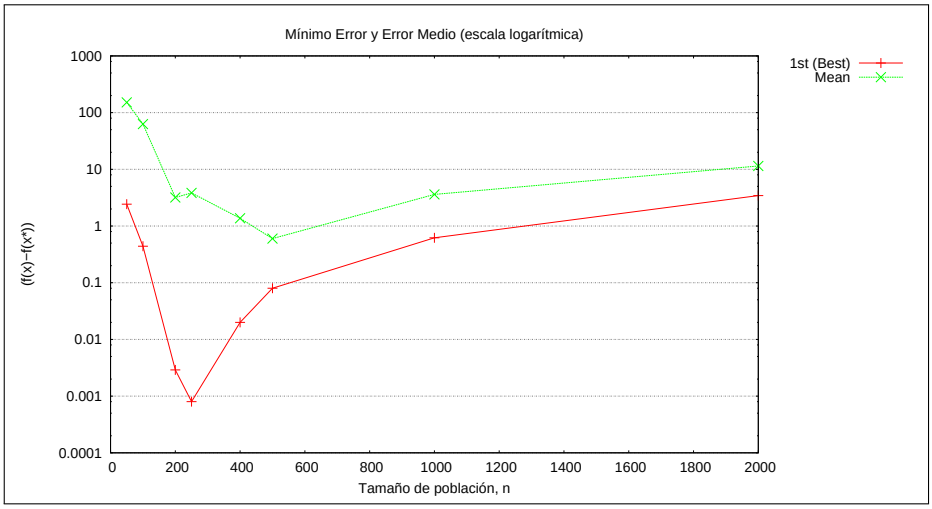

Figura 5.5: Mínimo error y error medio, en relación con el tamaño de población, en los estudios de 10.D.**.0005

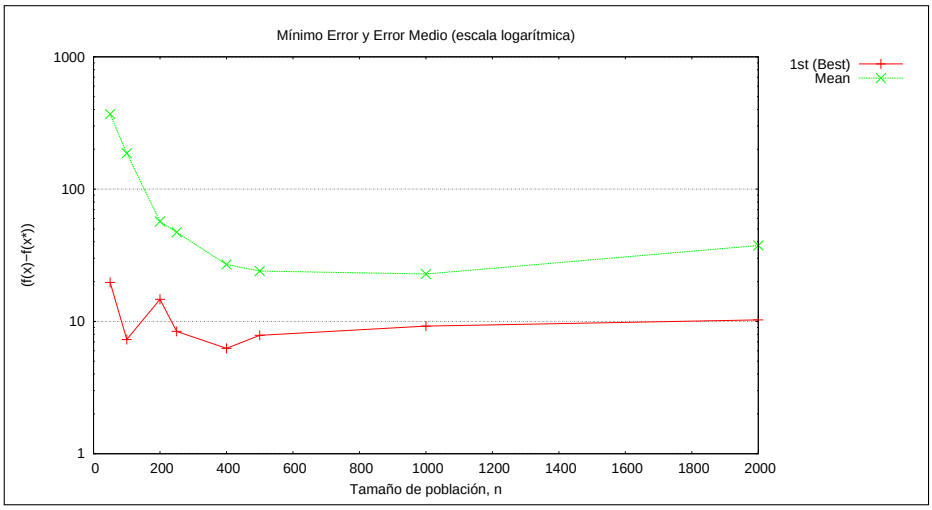

Figura 5.6: Mínimo error y error medio, en relación con el tamaño de población, en los estudio de 10.R.**.0005

Primeramente se han escogido los dos mejores tamaños de población de la ruleta, $n = 400$ y $n = 500$, ya que al ser la opción que peores resultados ofrece de las dos, preferimos partir con lo mejor de esta. Para d'Hondt, se ha escogido el mejor tamaño de población ($n = 250$), pero en cambio, con el fin de compartir al menos un tamaño de población la ruleta y d'Hondt, en vez de coger el segundo mejor, se tomó un tamaño de población bueno que coincidiera con alguno de los de la ruleta: $n = 400$.

5.4.1.2. Análisis del operador

Para los tamaños de población escogidos, $n = 400$ y $n = 500$, en la ruleta o $n = 250$ y $n = 400$, en el d'Hondt, se recogen en este párrafo los resultados más relevantes obtenidos con las probabilidades de mutación $Pm_a = 0{,}0005$, $Pm_b = 0{,}005$ y $Pm_c = 0{,}05$, en los estudios de los grupos de escenarios 10.D.**.**** y 10.R.**.****. Como ya se ha comentado, en el apéndice F se muestran todos los resultados.

1. Grupo de escenarios 10.D.250.**** y 10.D.400.****

   Los datos reflejados en la tabla 5.12, muestran un mejor comportamiento del algoritmo, en general, en las condiciones de baja complejidad ($d = 10$) y operador selección d'Hondt, para el tamaño de población $n = 250$ que para $n = 400$. No obstante el error medio cuando se escoge este tamaño de población ($n = 250$) y la probabilidad de mutación $Pm = 0{,}0005$, es peor que en los dos mejores casos de $n = 400$.

   En lo que respecta a la probabilidad de mutación, resulta evidente que los resultados obtenidos con las probabilidades baja y media ($Pm_a = 0{,}0005$ y $Pm_b = 0{,}005$) son mucho mejores que para la probabilidad alta ($Pm_b = 0{,}05$). De entre ellas, la probabilidad de mutación media, muestra un menor error medio, mientras que en la baja destacan los bajos errores mínimos obtenidos.

   Por otro lado, fijándonos en la evolución del algoritmo con respecto al número de ejecuciones (FES), en las gráficas 5.7 y 5.8, vemos como tanto para el mínimo error como para el error medio, del paso de 100 a 1000 evaluaciones y de 1000 a 10000 (finalización), cuando se ejecuta con $Pm_a = 0{,}0005$ y $Pm_b = 0{,}005$ (baja y media) siempre se produce una mejora notable en cada uno de los pasos. No es así para $Pm_c = 0{,}05$, que aparte de obtener una peor mejora de de 100 a 1000 que para las otras mutaciones, en el paso de 1000 a 10000 evaluaciones, mejora poco, o en

otras palabras, converge.

Los escenarios 10.D.250.0005 y 10.D.400.005 destacan, uno por su bajo error mínimo (bastante menor que el resto) y el otro por tener un buen valor de error medio sin renunciar a alcanzar bajos errores mínimos.

Pm	0,0005		0,005		0,05	
n	250	400	250	400	250	400
1st (Best)	0,0008	0,023	0,020	0,044	245,95	255,41
Mean	3,84	1,37	0,24	0,34	435,9	471,43

Tabla 5.12: Mínimo error y error medio, tras finalizar, en los estudios de 10.D.250.**** y 10.D.400.****

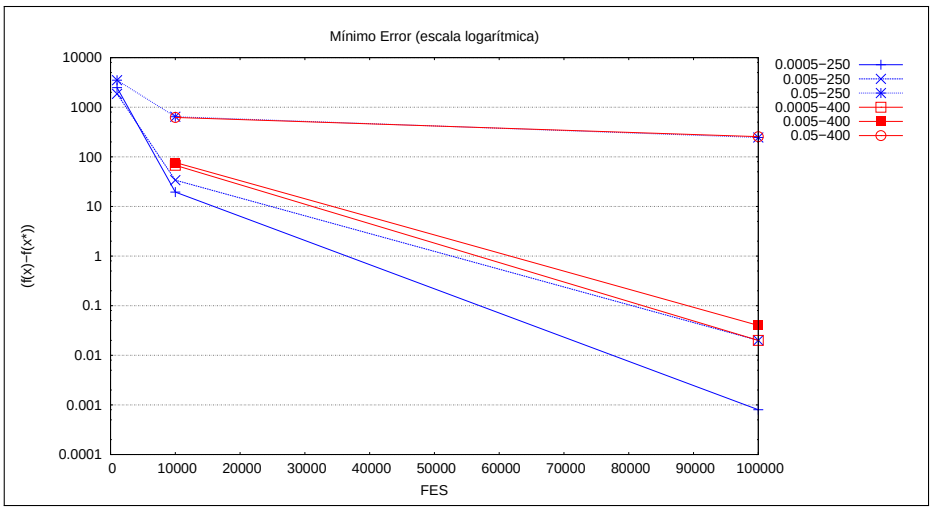

Figura 5.7: Mínimo error, según número de evaluaciones, en los estudios de 10.D.250.**** y 10.D.400.****

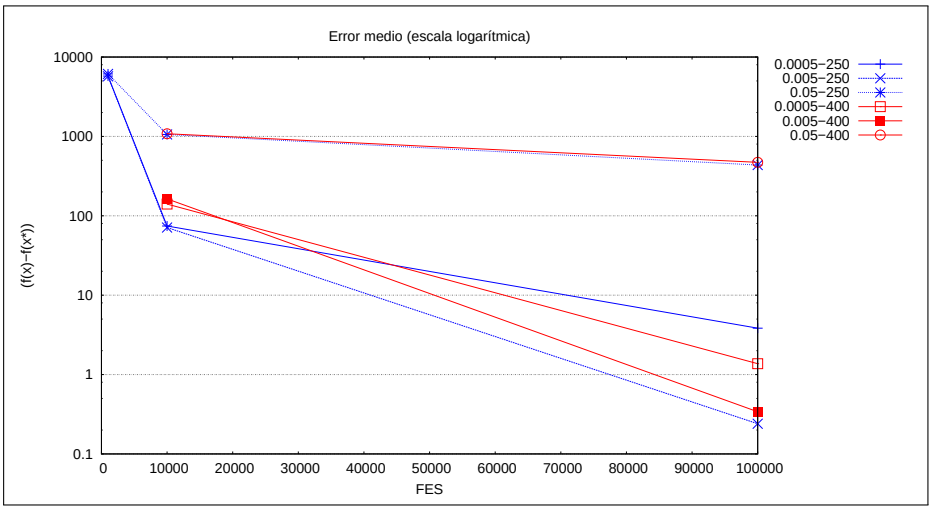

Figura 5.8: Error medio, según número de evaluaciones, en los estudios de 10.D.250.**** y 10.D.400.****

2. Grupo de escenarios 10.R.400.**** y 10.R.500.****

   Como se puede observar en los resultados recogidos en la tabla 5.13, referentes a las condiciones de baja complejidad ($d = 10$) y operador selección ruleta, ocurre que para las tres probabilidades de mutación, se obtiene siempre un error mínimo algo mejor para el tamaño de población $n = 400$ que para el tamaño de población $n = 500$. Sin embargo, en lo que se refiere al error medio, la cosa cambia completamente y es para el tamaño de población $n = 500$ cuando se alcanzan mejores valores que para $n = 400$, en los tres tamaños de población.

   Si atendemos a la probabilidad de mutación, según se aumenta la probabilidad (baja, media y alta) se obtienen peores valores, tanto para error medio como para el mínimo error.

   En lo referente a la evolución del algoritmo con respecto al número de evaluaciones (FES), mostrado en las gráficas 5.9 y 5.10, ocurre lo mismo que para el d'Hondt, es decir, para $Pm_c = 0{,}05$ en comparación con $Pm_a = 0{,}0005$ y $Pm_b = 0{,}005$ alcanza una peor mejora de de 100 a 1000 y en el paso de 1000 a 10000 evaluaciones, mejora muy poco (convergencia).

   A destacar 10.R.400.0005 y 10.R.500.0005, uno de ellos por su error

mínimo con un buen error medio y el otro por el error medio con un buen error mínimo.

Pm	0,0005		0,005		0,05	
n	400	500	400	500	400	500
1st (Best)	6,26	7,85	13,59	17,43	436,85	391,76
Mean	26,93	24,04	38,57	36,71	831,91	753,01

Tabla 5.13: Mínimo error y error medio, tras finalizar, en los estudios de 10.R.400.**** y 10.R.500.****

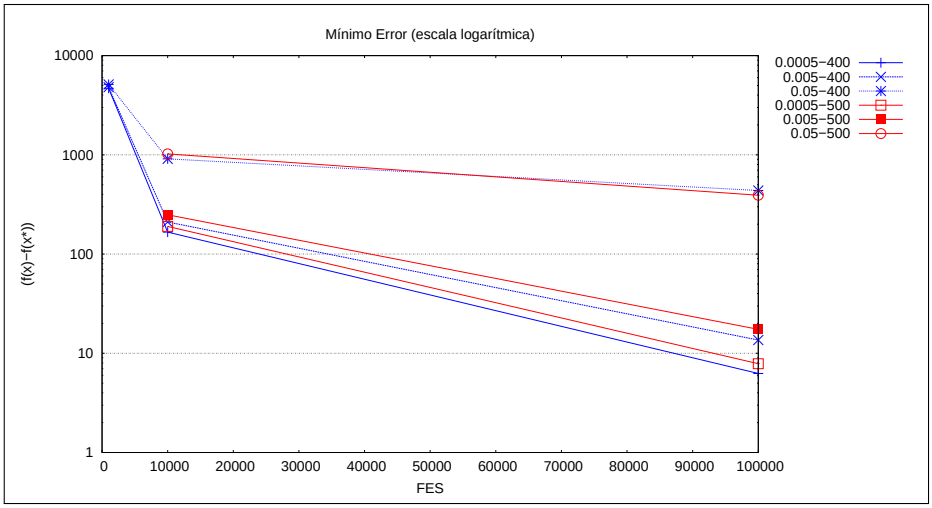

Figura 5.9: Mínimo error, según número de evaluaciones, en los estudios de 10.R.400.**** y 10.R.500.****

En el apartado del diseño del experimento, dentro de las estrategias a seguir (apartado 5.1.3), se contemplaba la posibilidad de añadir alguna probabilidad de mutación Pm a la experimentación, en caso de que, con algún valor intermedio entre Pm_a, Pm_b y Pm_c o con un valor superior a Pm_c, se intuyera que se pudieran obtener mejores resultados. Como se ha comentado en estos dos grupos de escenarios, 10.D.**.0005 y 10.R.**.0005, según se aumenta la probabilidad (baja, media y alta) se obtienen peores valores, por lo que podemos decir que estudios con probabilidades intermedias o muy altas no nos aportarían más información.

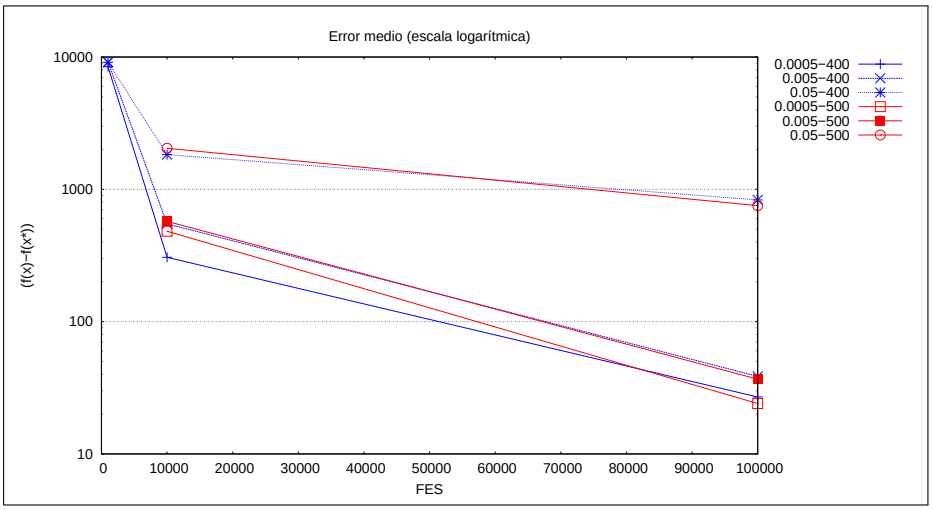

Figura 5.10: Error medio, según número de evaluaciones, en los estudios de 10.R.400.**** y 10.R.500.****

5.4.1.3. Comparativa entre d'Hondt y ruleta para $d = 10$

A continuación se comparan los resultados de los estudios en relación al algoritmo con operador selección d'Hondt, escenarios 10.D.250.0005 y 10.D.400.0005 y con operador selección ruleta, escenario 10.R.400.0005, para la complejidad $d = 10$, según los 5 criterios del PDEC-05.

De entre los cuatro tamaños de población estudiados en el análisis del operador, se ha escogido el mejor para el d'Hont: $n = 250$ (escenario 10.D.250.0005) y el mejor para la ruleta: $n = 400$ (escenario 10.R.400.0005). Además, como ya se ha comentado en la determinación del tamaño de población, se ha incluido una comparación con el mismo tamaño de población, para los dos operadores: $n = 400$ (escenarios 10.D.400.0005 y 10.R.400.0005), favorable a la ruleta, al ser su mejor resultado.

1. Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) en la finalización, para cada ejecución (dada por Ter_Err o Max_FES).

   Tal como se puede observar en las gráficas (con la escala logarítmica) 5.11 y 5.12, comparando el escenario con operador selección ruleta, 10.R.400.0005 y el mejor escenario con operador selección d'Hondt, 10.D.250.0005, vemos que para el error medio, el estudio con el operador selección d'Hondt resultó mucho mejor que para el operador selección ruleta, tanto para $FES = 1e4$ como para $FES = 1e5$, aunque eso sí, con

una mejora muy similar en el paso de 1e4 a 1e5. Si atendemos al menor error (ver además la tabla 5.14), también más fácil de observar con escala logarítmica, la mejor ejecución con d'Hondt es muy superior al de ruleta, sobre todo para $FES = 1e5$, con una diferencia de orden de magnitud de 5 (0,0008 frente a 6,26), ya que para $FES = 1e4$ no lo es tanto, ya que la diferencia de orden 1 (19,48 frente a 166,31).

Si analizamos el escenario con operador selección ruleta, 10.R.400.0005, con su homólogo el d'Hondt 10.D.400.0005 (mismo tamaño de población, gráficas 5.13 y 5.14 y tabla 5.14), vemos que para el error medio y $FES = 1e5$ es incluso mejor que el d'Hondt y $n = 250$, aunque para $FES = 1e4$ sería similar. Con respecto al menor error, la diferencia para 1e5 y 1e4, se ha reducido algo pero sigue siendo muy grande: 0,02 frente a 6,26 (diferencia en el orden de 2) y 67,63 frente a166,31 (diferencia de orden 1) respectivamente.

El mejor "registro" lo obtiene por lo tanto el escenario 10.D.250.0005 con el error mínimo 0,0008 (del orden de 10e-4).

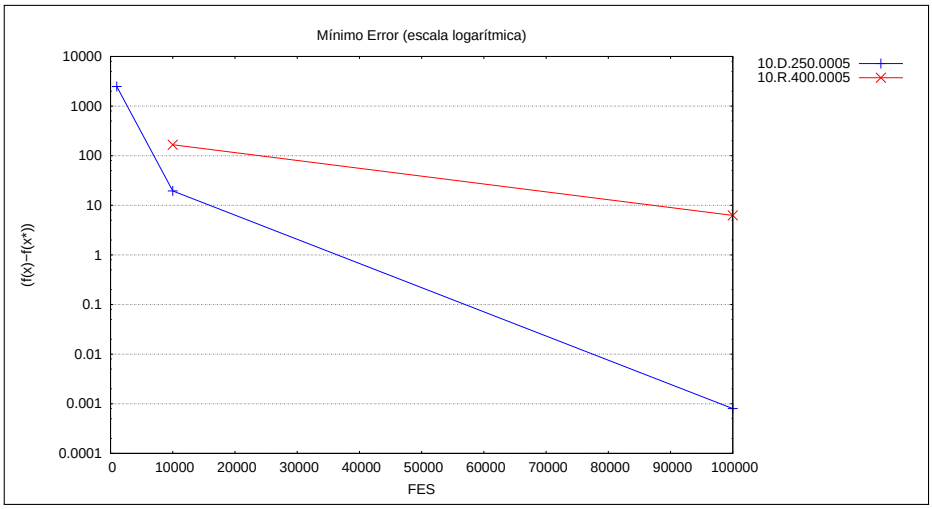

Figura 5.11: Mínimo error, según número de evaluaciones. Comparativa escenarios 10.D.250.0005 y 10.R.400.0005

FES \ Escenario		10.D.250.0005	10.R.400.0005	10.D.400.0005
1e3	1st (Best)	2484,74		
	7st	4455,26		
	13st (Median)	5179,97		
	19st	6939,94		
	25st (Worst)	11000,54		
	Mean	5763,02		
	Std	1938,81		
1e4	1st (Best)	19,48	166,31	67,63
	7st	55,18	212,44	95,81
	13st (Median)	74,95	299,58	129,93
	19st	92,8	354,17	169,98
	25st (Worst)	163,3	577,21	301,15
	Mean	74,4	306,3	140,62
	Std	33,6	112,42	60,05
1e5 (end)	1st (Best)	0,0008	6,26	0,023
	7st	0,14	11,52	0,16
	13st (Median)	0,69	21,81	0,399
	19st	3,73	35,43	0,93
	25st (Worst)	30,89	83,09	7,84
	Mean	3,84	26,93	1,37
	Std	7,15	19,55	2,04

Tabla 5.14: Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) en la finalización, para cada ejecución. Comparativa escenarios 10.D.**.**** y 10.R.**.****

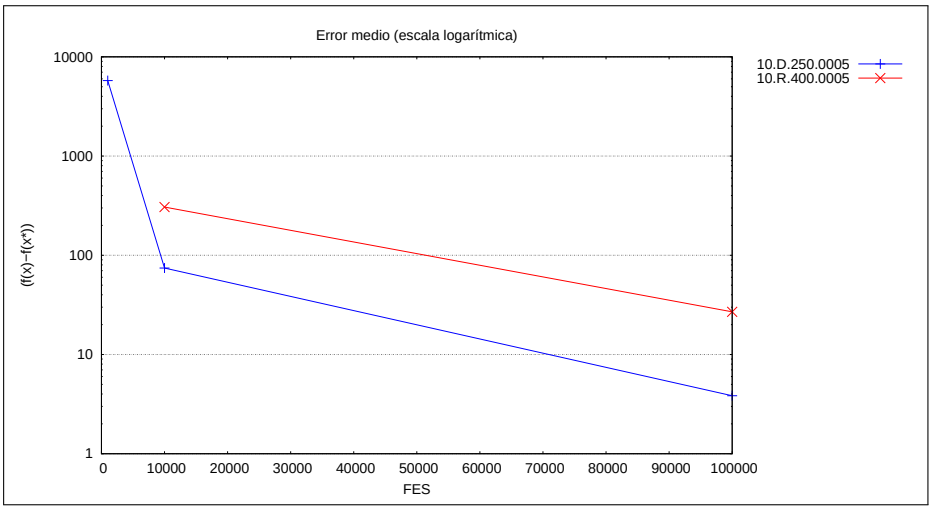

Figura 5.12: Error medio, según número de evaluaciones. Comparativa escenarios 10.D.250.0005 y 10.R.400.0005

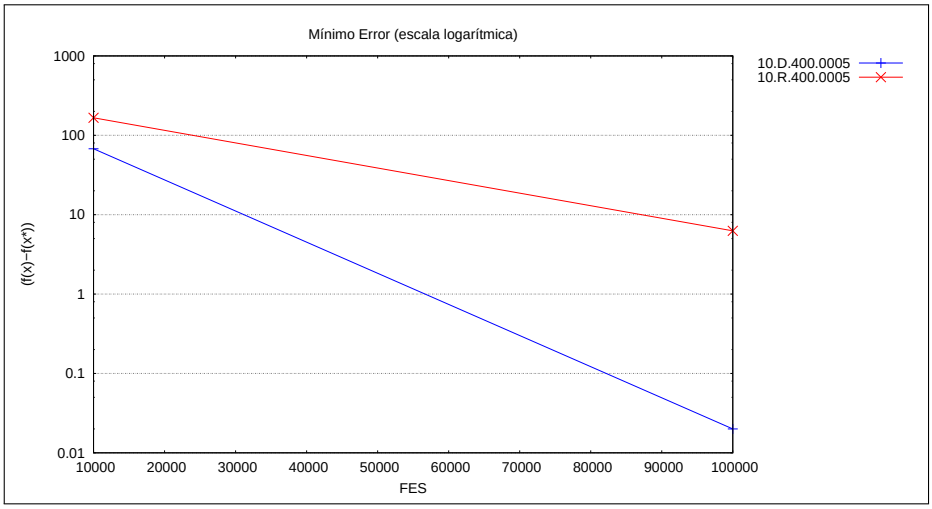

Figura 5.13: Mínimo error, según número de evaluaciones. Comparativa escenarios 10.D.400.0005 y 10.R.400.0005

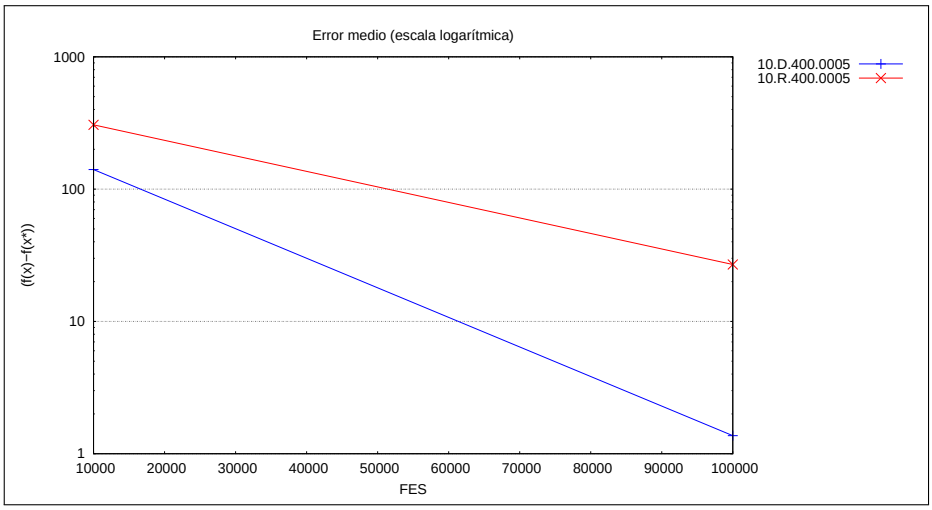

Figura 5.14: Error medio, según número de evaluaciones. Comparativa escenarios 10.D.400.0005 y 10.R.400.0005

2. Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el nivel de precisión. Ratio de éxito y rendimiento de éxito.

   Para el apartado de número de evaluaciones necesario para alcanzar una precisión determinada, se obtienen distintos resultados dependiendo del grado de precisión. Es importante aclarar que el valor MEDIO, ya que se computa sobre el número de veces que se alcanza el nivel de precisión puede llevar a error. Por ejemplo puede resultar que para un tamaño de población que solo alcance la precisión una o dos veces, pero con un número bajo de FES, contrasta con que para otro tamaño de población se alcance muchas más veces, con valores tan buenos como los del primero, pero que un par de malos resultados desvirtúen la media. Por ello solo vamos a analizar los FES alcanzados y será en los ratios, que contrastan FES y número de veces que se alcanza, los que nos aclararán la calidad del algoritmo.

   A partir de los valores de la tabla 5.15, se puede ver que para el valor alto de precisión, que es 1e-1, solamente los escenarios con operador selección d'Hondt logran alcanzar esta marca para algunos estudios. Si queremos que el escenario con el operador selección ruleta alcance una precisión determinada, nos tenemos que ir a precisión 10.

   Para el caso de precisión 10, el mejor escenario con d'Hondt, 10. D.250.0005, muestra un menor número de FES para alcanzar este valor de precisión de 12500 FES frente a los 55600 FES que necesita 10.R.400.0005. El ratio de éxito también es mucho mayor, 0,88 para el estudio con el operador d'Hond, frente a 0,2 con la ruleta y el ratio de rendimiento de éxito (success performance) es menor (mejor) con el d'Hondt (29777,89 FES) que con la ruleta (352800 FES). Si comparamos el escenario con operador selección ruleta, 10.R.400.0005, y aquel con el d'Hondt y con el mismo tamaño de población $n = 400$, tenemos que la diferencia en cuanto al menor número de FES para alcanzar al precisión de 10 es menor (19200 FES para el d'Hondt y 55600 FES para la ruleta), pero en cambio el ratio de éxito para el d'Hondt, es ahora de 1 (todas las ejecuciones alcanzan la precisión) y el rendimiento de éxito sigue siendo mejor (352800 frente ahora 35632 FES).

   Con un nivel de precisión inferior (de 100), vemos en cuestión de ratio de éxito que tanto los estudios con d'Hondt como el estudio con la ruleta, alcanzan el mencionado valor de error. En cambio, para el escenario con operador selección ruleta, el menor número de FES para alcanzar la

precisión y el ratio de rendimiento de éxito, siguen siendo peor que para los escenarios con operador selección ruleta, aunque esta vez no con tanta diferencia, tal como se puede ver en la tabla 5.15.

Atendiendo a las gráficas, 5.15, 5.16 y 5.17 para la comparativa escenarios 10.D.250.0005 y 10.R.400.0005 y 5.18, 5.19 y 5.20 para la comparativa escenarios 10.D.400.0005 y 10.R.400.0005, vemos que siempre existe un mejor comportamiento con el operador selección d'Hondt que con la ruleta, sobre cuanto mayor es el nivel de precisión (valores más pequeños). Si la precisión es baja (valores más altos), los resultados entre los dos operadores se aproximan, pero siempre siendo mejores los del operador d'Hondt.

n	1st (Best)	7st	13st (Median)	19st	25st (Worst)	Mean	Std	Nivel de precisión	Success Rate	Success Perform.
10.D.250.0005	-	-	-	-	-	-	-	1,00E-006	-	-
10.R.400.0005	-	-	-	-	-	-	-	1,00E-006	-	-
10.D.400.0005	-	-	-	-	-	-	-	1,00E-006	-	-
10.D.250.0005	56250	-	-	-	-	69200	11289,38	1,00E-001	0,2	346000
10.R.400.0005	-	-	-	-	-	-	-	1,00E-001	-	-
10.D.400.0005	91200	-	-	-	-	94640	2906,54	1,00E-001	0,2	473200
10.D.250.0005	23750	46500	73750	-	-	53125	18148,36	1,00E+000	0,56	94866,07
10.R.400.0005	-	-	-	-	-	-	-	1,00E+000	-	-
10.D.400.0005	51200	59200	73200	97200	-	69347,37	15126,85	1,00E+000	0,76	91246,54
10.D.250.0005	12500	22250	25000	31250		26204,55	10851,99	1,00E+001	0,88	29777,89
10.R.400.0005	55600	-	-	-	-	70560	11550,24	1,00E+001	0,2	352800
10.D.400.0005	19200	27600	37200	41200	57600	35632	10201,91	1,00E+001	1	35632
10.D.250.0005	5000	6750	8500	9500	12750	8320	2065,99	1,00E+002	1	8320
10.R.400.0005	12000	15600	17200	27600	36400	20768	7797,37	1,00E+002	1	20768
10.D.400.0005	7600	10000	11600	14000	18800	12000	2744,69	1,00E+002	1	12000

Tabla 5.15: Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión, ratio de éxito y rendimiento de éxito. Comparativa escenarios 10.D.**.**** y 10.R.**.****

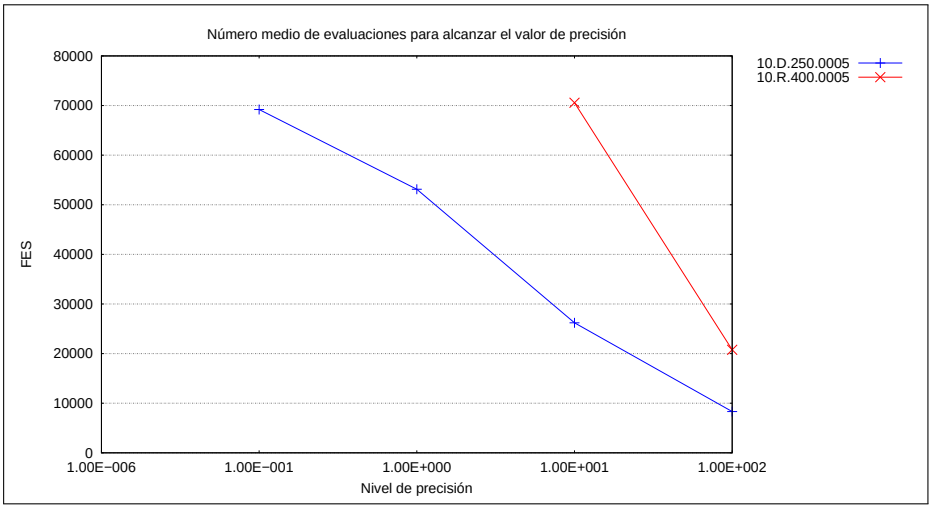

Figura 5.15: Número medio de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 10.D.250.0005 y 10.R.400.0005

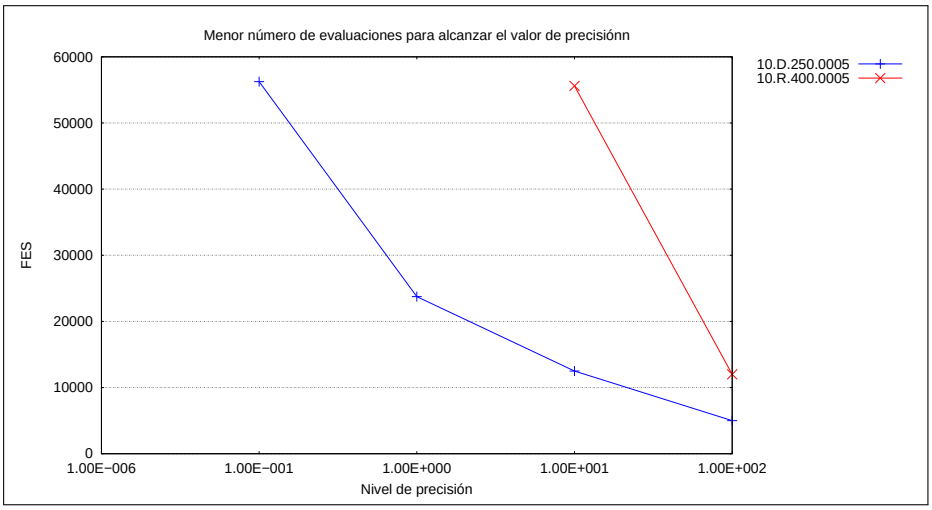

Figura 5.16: Menor número de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 10.D.250.0005 y 10.R.400.0005

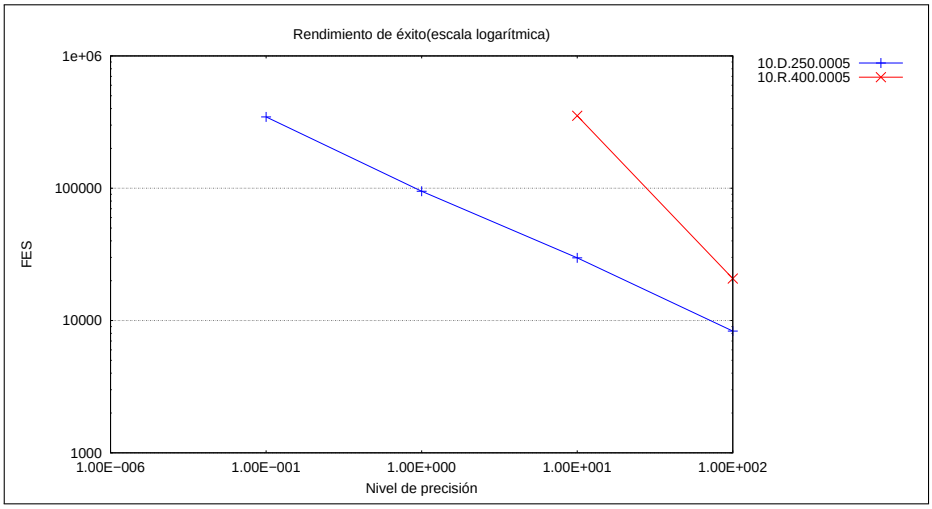

Figura 5.17: Rendimiento de éxito. Comparativa escenarios 10.D.250.0005 y 10.R.400.0005

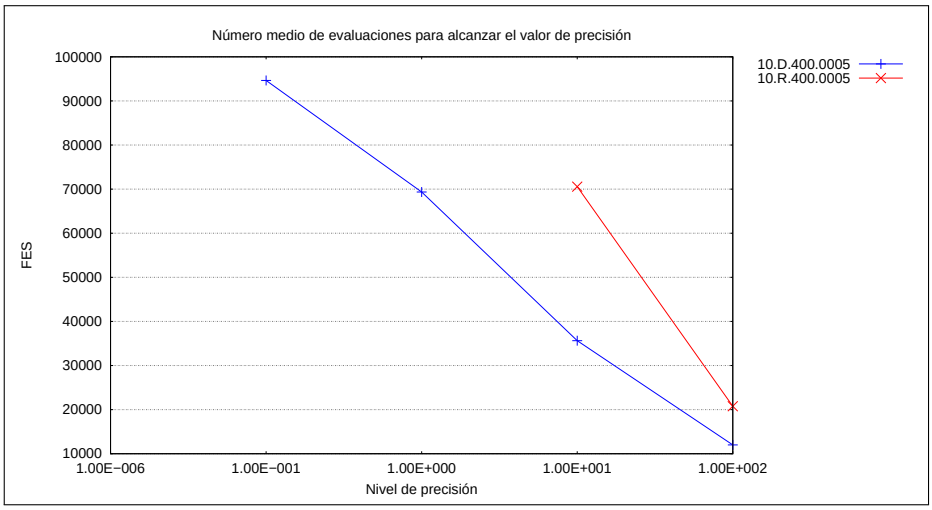

Figura 5.18: Número medio de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 10.D.400.0005 y 10.R.400.0005

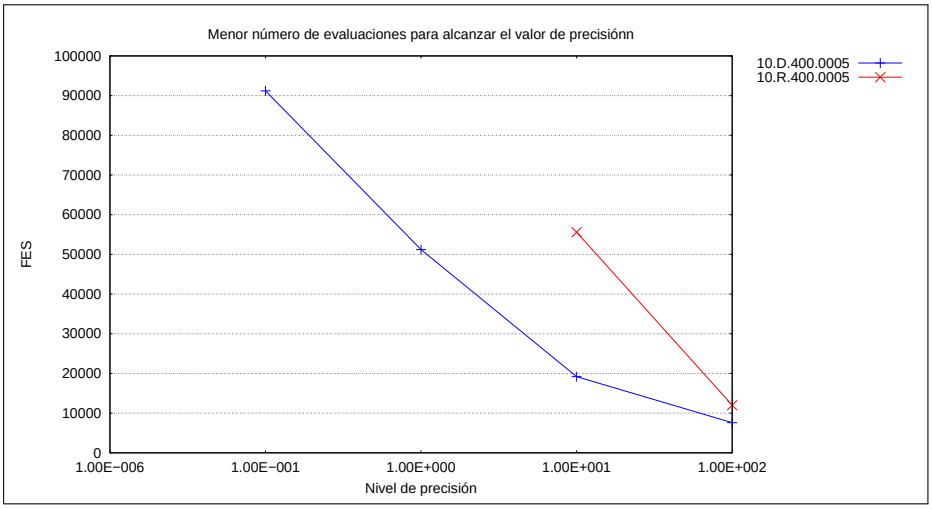

Figura 5.19: Menor número de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 10.D.400.0005 y 10.R.400.0005

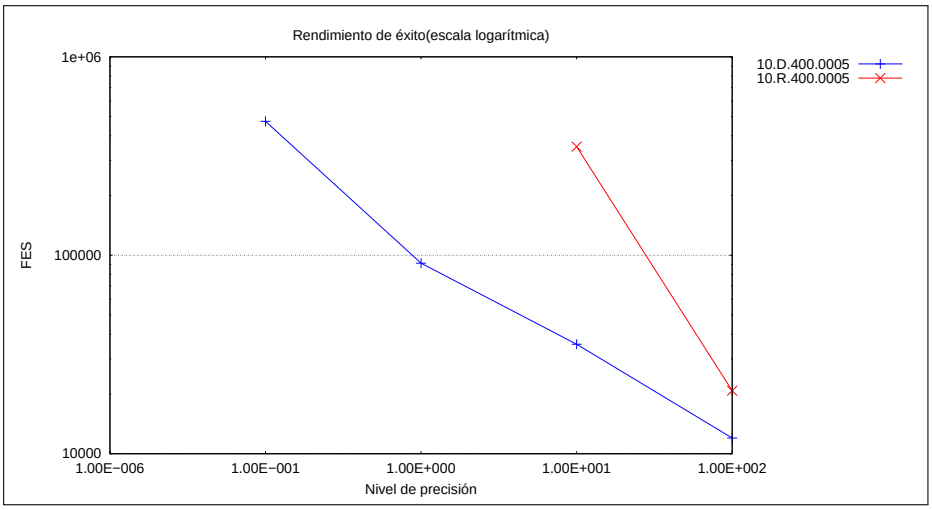

Figura 5.20: Rendimiento de éxito. Comparativa escenarios 10.D.400.0005 y 10.R.400.0005

3. Gráficas de convergencia.

   Tal como se observa en las gráficas de convergencia 5.21 y 5.22, el mejor escenario con d'Hondt, 10.D.250.0005, muestra un cierto conjunto de ejecuciones que convergen frente a la gráfica de convergencia del escenario con ruleta, 10.R.400.0005 en el que se podría decir que convergen todos. Si atendemos al homólogo de este escenario en cuanto a tamaño de población, pero con d'Hondt, 10.D.400.0005 (gráfica 5.22), solo algunos individuos convergen frente a la ruleta que como se ha dicho, convergen todos.

   Lo más interesante, si comparamos las gráficas de convergencia (figura 5.24), es que para el escenario con el d'Hondt, 10.D.250.0005, y la ruleta 10.R.400.0005, vemos que casi todas las ejecuciones para el d'Hondt muestran un menor logaritmo del error que con la ruleta y solo algunas de mejores ejecuciones de la ruleta logran mejorar a algunas de las peores del d'Hondt.

   Solo si comparamos el escenario con el d'Hondt, 10.D.400.0005, y la ruleta 10.R.400.0005 (5.25), es decir, con el mismo tamaño de población, entonces se produce un "solapamiento", algo mayor, sobre todo en las primeras evaluaciones. Aún así, el d'Hondt sigue mostrando superioridad con respecto a la ruleta e incluso, para evaluaciones, próximas a 1e5, solo la mejor ejecución de la ruleta resulta mejor o igual que al peor ejecución del d'Hondt (con $n = 400$).

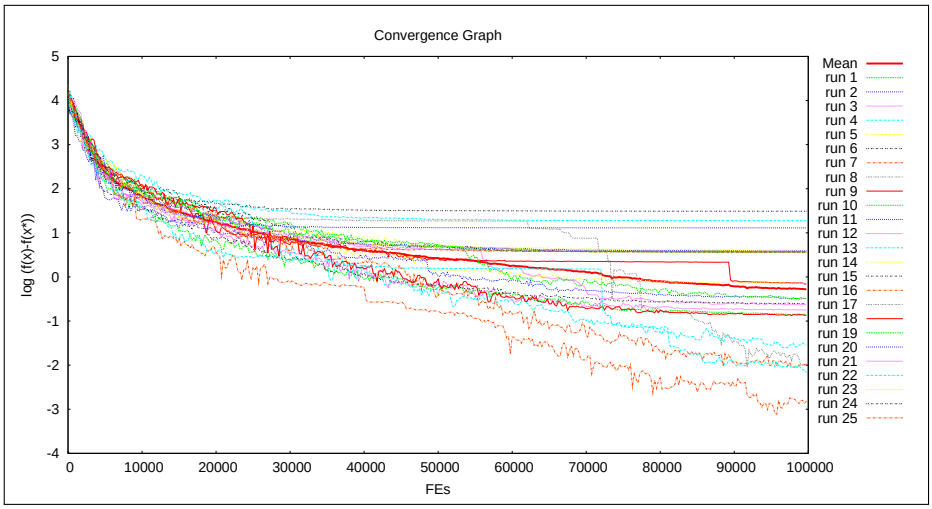

Figura 5.21: Convergencia. Escenario 10.D.250.0005

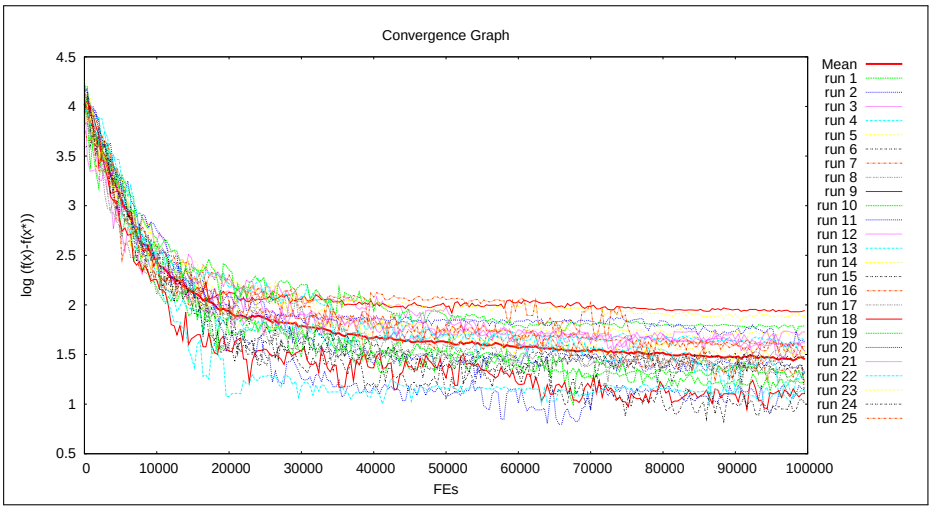

Figura 5.22: Convergencia. Escenario 10.R.400.0005

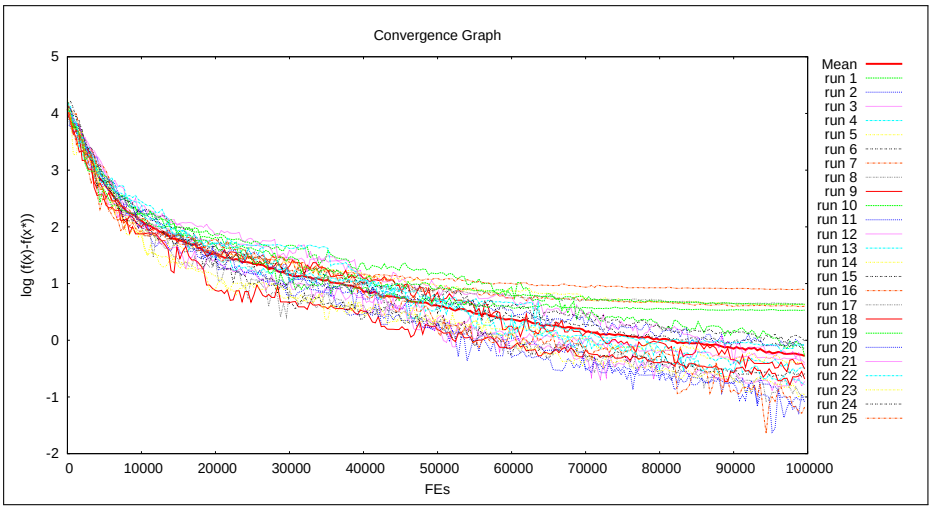

Figura 5.23: Convergencia. Escenario 10.D.400.0005

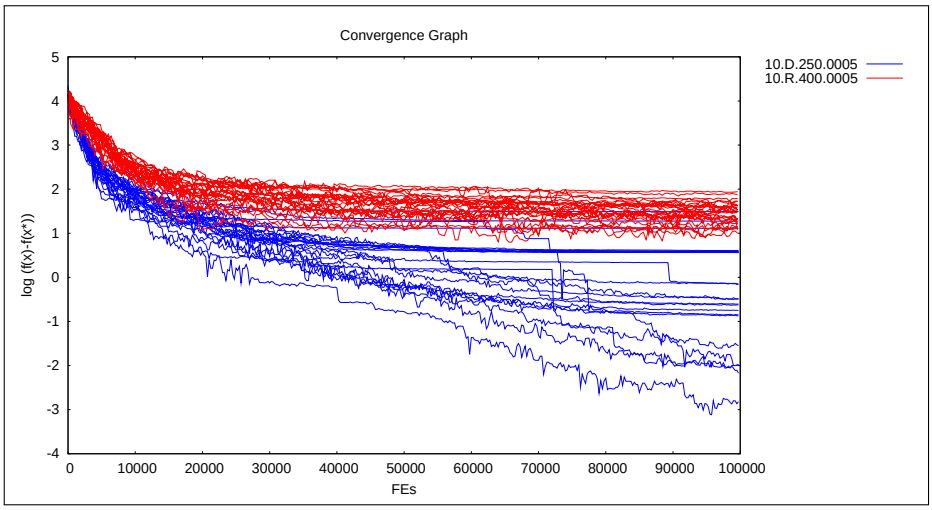

Figura 5.24: Convergencia. Comparativa escenarios 10.D.250.0005 y 10.R.400.0005

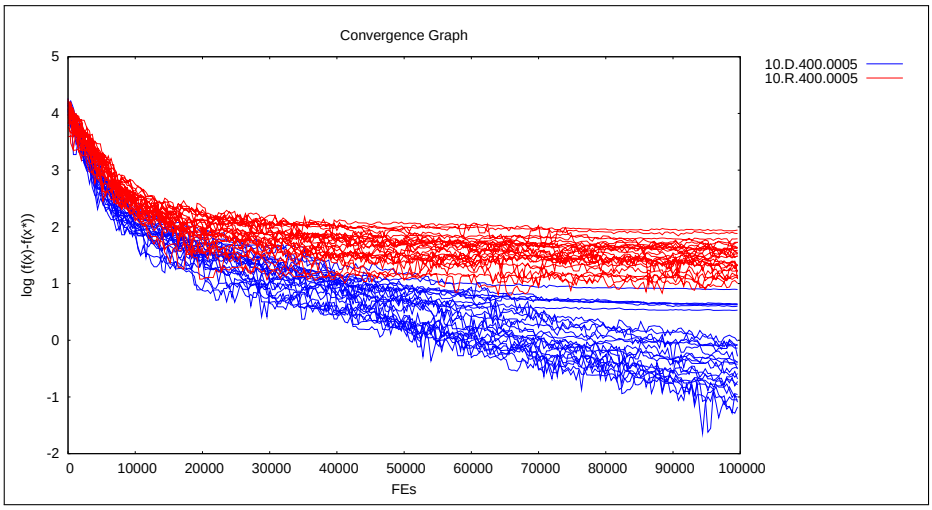

Figura 5.25: Convergencia. Comparativa escenarios 10.D.400.0005 y 10.R.400.0005

4. Algorithm Complexity.

   Atendiendo a los valores de la tabla, no podemos sacar ninguna conclusión, salvo que los valores de complejidad del algoritmo (Algorithm Complexity $= \frac{\hat{T_2}-T_1}{T_0}$, apéndice D) son muy similares.

Escenario	T0	T1	T2'	(T2'-T1)/T0
10.D.250.0005	0,18	1,77	21,56	109,97
10.R.400.0005			21,26	108,27
10.D.400.0005			21,49	109,56

Tabla 5.16: Algorithm Complexity- computing_time_t2. Comparativa escenarios 10.D.250.0005, 10.R.400.0005 y 10.D.400.0005

5.4.2. Resultados alta complejidad ($d = 50$)

5.4.2.1. Determinación del tamaño de población

Con la alta complejidad se han realizado estudios, con los siguientes tamaños de población n: 250, 400, 500, 1000, 2000 y 2500 para el grupo de escenarios 50.D.**.0005 y 500, 1000, 2000, 2500 y 5000 en 50.R.**.0005. Tanto los datos de mínimo error[2] y error medio del estudio de 5e5 (al final de la ejecución), para cada tamaño de población n en los dos casos (operadores selección), se muestran en la siguiente tabla 5.17. Además, las gráficas 5.5 y 5.6 muestran la relación entre el error y el tamaño de población.
Partiendo de los resultados anteriores, con la complejidad $d = 50$, para el d'Hondt los mejores tamaños de población son $n = 500$, $n = 1000$ y $n = 2000$. En el caso de la ruleta, tenemos que los mejores resultados se alcanzan con $n = 1000$, $n = 2000$ y $n = 2500$. Siguiendo la misma estrategia que para $d = 10$, determinaremos primeramente los dos mejores tamaños de población para la ruleta y con estos valores y los resultados obtenidos en el d'Hondt escogeremos los de este operador.

Para la ruleta la experimentación con una población de $n = 1000$ individuos es la que mejor resultado final obtiene, por lo que este tamaño queda escogido. Entre las siguientes opciones, $n = 2000$ y $n = 2500$, si fuéramos rigurosos, escogeríamos $n = 2500$ ya que el mejor resultado es algo mejor que el de $n = 2000$, pero si observamos la media y la dispersión de $n = 2000$ y tenemos en cuenta que este tamaño de población está disponible para la ley d'Hont,

[2]El mínimo error se trata, de las 25 ejecuciones del estudio, del primer error de la función: el mejor (ver apartado 3.2.1)

d'Hondt							
Tamaño de población	250	400	500	1000	2000	2500	
1st (Best)	234,16	167,66	39,96	38,64	39,82	105,8	
Mean	907,67	611,47	427,32	209,55	127,09	195,7	
Ruleta							
Tamaño de población			500	1000	2000	2500	5000
1st (Best)			549,19	260,46	449,88	405,02	996,23
Mean			941	658,48	636,05	687,16	1292,96

Tabla 5.17: Mínimo error y error medio, tras finalizar, en los estudios de 50.D.**.0005 y 50.R.**.0005

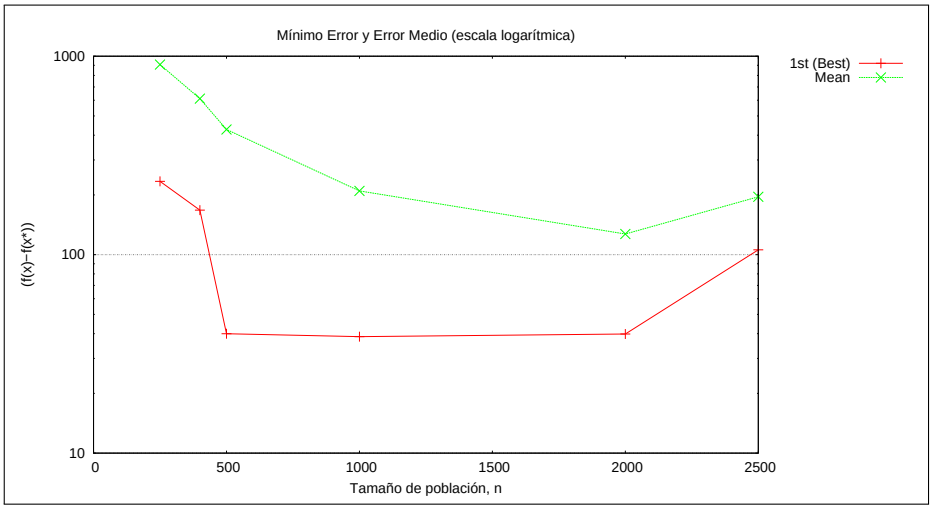

Figura 5.26: Mínimo error y error medio, en relación con el tamaño de población, en los estudios de 50.D.**.0005

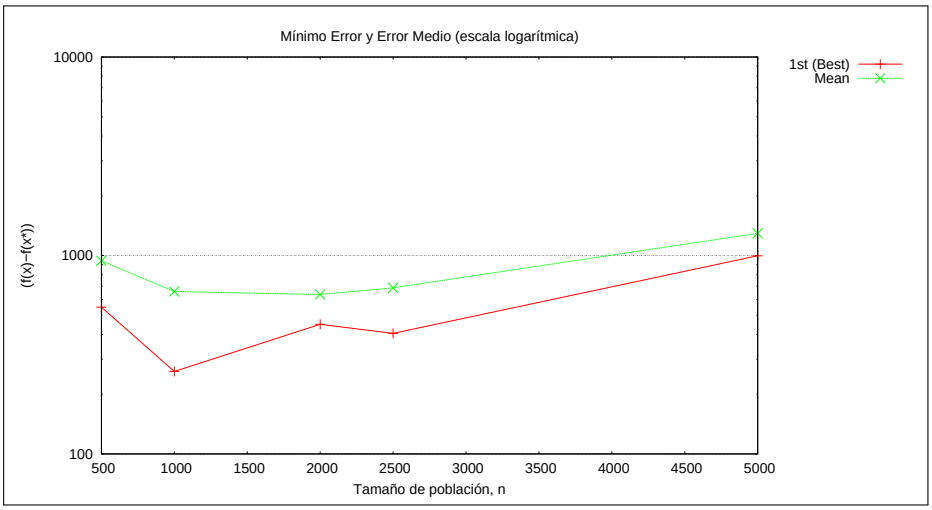

Figura 5.27: Mínimo error y error medio, en relación con el tamaño de población, en los estudio de 50.R.**.0005

finalmente nos quedamos con $n = 2000$.

En lo que se refiere al d'Hondt, para $n = 1000$ se obtiene el mejor resultado, y dado que este tamaño también ha sido escogido para la ruleta, queda claro que este será el primer tamaño de población elegido para este operador. La duda viene ahora entre $n = 2000$ y $n = 500$. El mínimo error al final de la ejecución es casi el mismo, pero el error medio es bastante más bajo para $n = 2000$. Además, como hemos visto en el párrafo anterior, $n = 2000$ también ha sido escogido para la ruleta, por lo que nos declinaremos por este tamaño de población.

Por lo tanto los tamaños de población escogidos para el d'Hondt y para la ruleta coinciden ($n = 1000$ y $n = 2000$), lo que resulta de gran ventaja a la hora de comparar resultados ya que las condiciones son exactamente las mismas tanto para un operado como para el otro.

5.4.2.2. Análisis del operador

Partiendo de los tamaños población escogidos, $n = 1000$ y $n = 2000$ (tanto para la ruleta como para el d'Hondt), a continuación se muestran los resultados más relevantes obtenidos con las probabilidades de mutación $Pm_a = 0{,}0005$, $Pm_b = 0{,}005$ y $Pm_c = 0{,}05$, en los estudios de los grupos de escenarios 50.D.**.**** y 50.R.**.****. Así mismo, en el apéndice F se muestran todos los resultados.

1. Grupo de escenarios 50.D.1000.**** y 50.D.2000.****

Atendiendo a los datos mostrados en las gráficas 5.28 y 5.29 se aprecia, en general, un mejor comportamiento, en las condiciones de alta complejidad ($d = 50$) y operador selección d'Hondt, para el tamaño de población $n = 1000$ que para $n = 2000$. Incluso se aprecia para las probabilidades de mutación baja y media ($Pm_a = 0{,}0005$ y $Pm_b = 0{,}005$), como tanto el error medio como el error mínimo son menores.

Sin embargo si observamos más detenidamente los datos mostrados en la tabla 5.18 y atendemos a la finalización ($FES = 50000$) en las gráficas mencionadas, vemos que tal superioridad no llega a ser tan aventajada. En primer lugar, para una probabilidad baja ($Pm_a = 0{,}0005$), el mejor resultado es menor para $n = 1000$ que para $n = 2000$, lo que nos muestra que este es el mejor tamaño; sin embargo la media sale bastante más perjudicada que en $n = 2000$. Analizando la probabilidad de mutación alta ($Pm_c = 0{,}05$) incluso el mejor registro con $n = 1000$ es peor que con $n = 2000$, salvándose en el error medio donde si es ligeramente mejor. Solamente para la probabilidad de mutación media ($Pm_b = 0{,}005$) es con el tamaño de población $n = 1000$, donde en ambos casos (valor mínimo y valor medio) el algoritmo es mejor que con $n = 2000$.

A pesar de estos últimos resultados, en líneas generales el algoritmo con el tamaño de población $n = 1000$ resulta mejor que con $n = 2000$, en particular para $Pm_a = 0{,}0005$.

Pm	0,0005		0,005		0,05	
n	1000	2000	1000	2000	1000	2000
1st (Best)	38,64	39,82	722,28	870,38	41126,71	39342,53
Mean	209,55	127,09	1031,02	1153,88	47206,02	47255,65

Tabla 5.18: Mínimo error y error medio, tras finalizar, en los estudios de 50.D.1000.**** y 50.D.2000.****

2. Grupo de escenarios 50.R.1000.**** y 50.R.2000.****

   Al igual que para 50.D.**.****, los datos reflejados en la tabla 5.19 y las gráficas 5.30 y 5.31, muestran un mejor comportamiento del algoritmo, en general (en las condiciones de alta complejidad ($d = 50$) y operador selección ruleta), para el tamaño de población $n = 1000$ que para $n = 2000$.

   No obstante atendiendo de nuevo a los resultados en la finalización ($FES = 50000$) vemos como no siempre es así. Para la probabilidad de

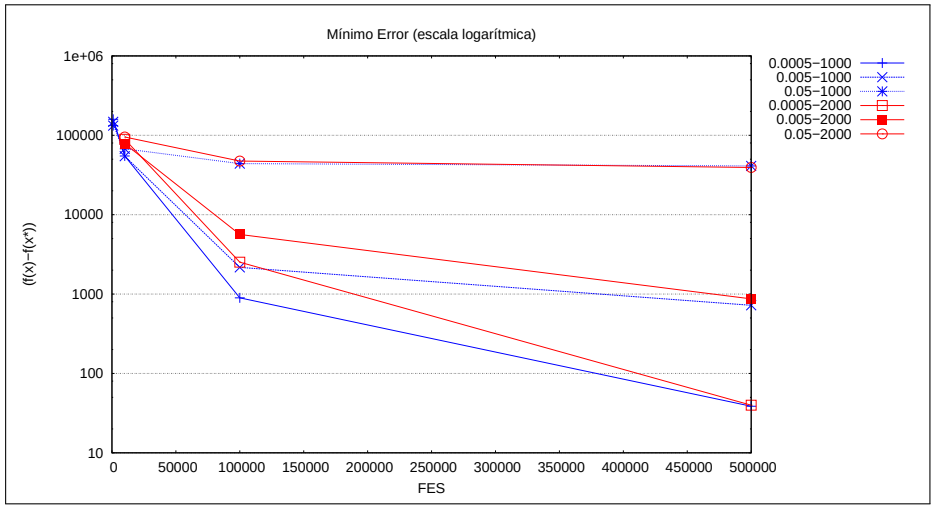

Figura 5.28: Mínimo error, según número de evaluaciones, en los estudios de 50.D.1000.**** y 50.D.2000.****

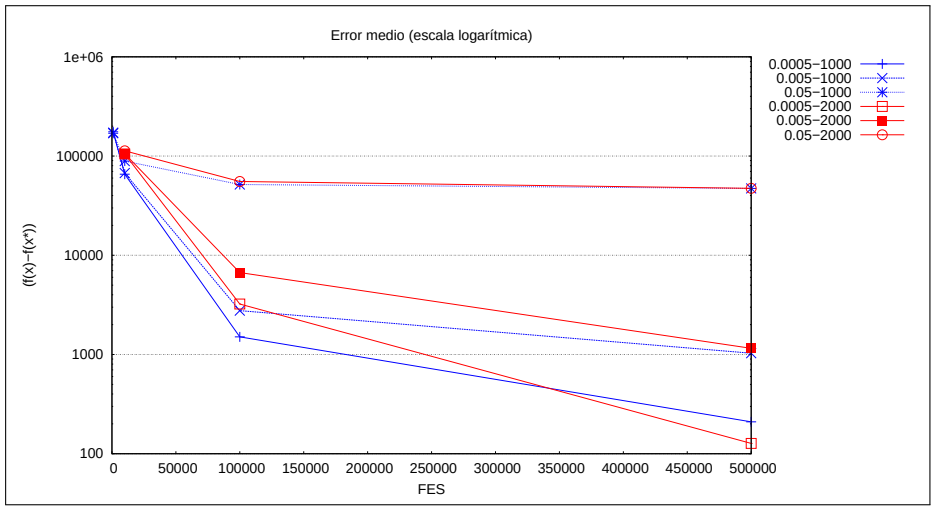

Figura 5.29: Error medio, según número de evaluaciones, en los estudios de 50.D.1000.**** y 50.D.2000.****

mutación media $Pm_b = 0{,}005$ el error mínimo es ligeramente peor con $n = 1000$ que con $n = 2000$, sin embargo el error mínimo si es bastante menor con $n = 10000$. Si nos fijamos en la probabilidad de mutación alta $Pm_c = 0{,}05$, vemos que los resultados son similares con una pequeña ventaja para $n = 2000$.

En cambio, en lo que respecta a la probabilidad de mutación alta $Pm_a = 0{,}0005$, tanto para el error mínimo como para el error medio es el tamaño de población $n = 1000$ abultadamente menor que $n = 2000$, lo que resulta evidente que el algoritmo con el tamaño de población con este tamaño de población ($n = 1000$) se comporta mejor.

Pm	0,0005		0,005		0,05	
n	1000	2000	1000	2000	1000	2000
1st (Best)	260,46	449,88	2265,27	1982,57	48741,95	46752,63
Mean	658,48	636,05	2682,62	3066,82	57218,33	57049,23

Tabla 5.19: Mínimo error y error medio, tras finalizar, en los estudios de 50.R.1000.**** y 50.R.2000.****

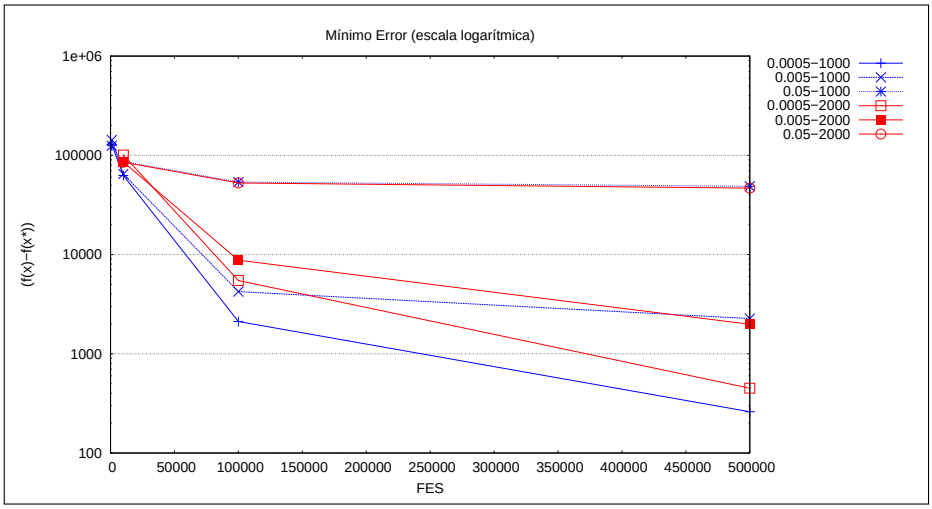

Figura 5.30: Mínimo error, según número de evaluaciones, en los estudios de 50.R.1000.**** y 50.R.2000.****

Como se mostró en las estrategias a seguir (apartado 5.1.3), si algún valor intermedio entre Pm_a, Pm_b y Pm_c o con un valor superior a Pm_c, se intuyera que se pudieran obtener mejores resultados, añadiría alguna probabilidad de mutación

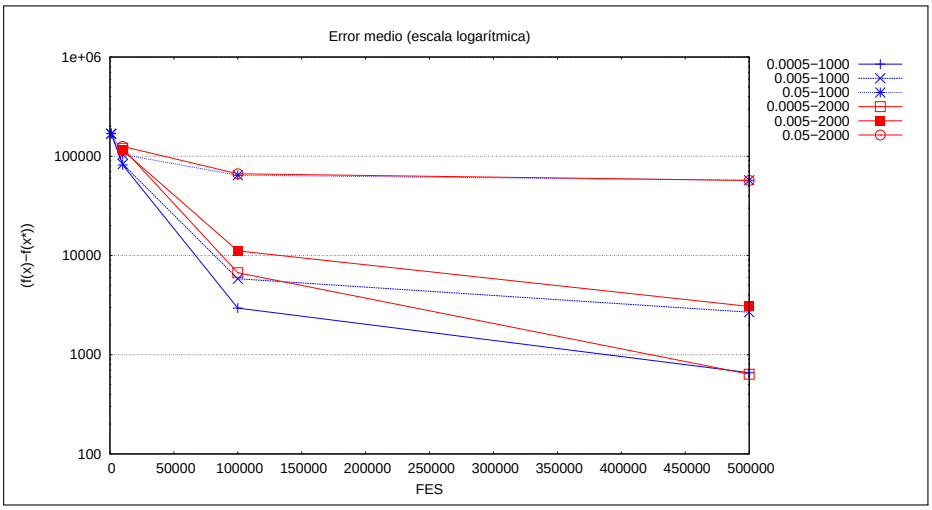

Figura 5.31: Error medio, según número de evaluaciones, en los estudios de 50.R.1000.**** y 50.R.2000.****

Pm a la experimentación. Como se ve en las anteriores tablas de resultados 5.18 y 5.19, estos dos grupos de escenarios, 50.D.**.0005 y 50.R.**.0005, según se aumenta la probabilidad (baja, media y alta) se obtienen peores valores, por lo que al igual que para $d = 10$ podemos decir que estudios con probabilidades intermedias o muy altas no nos aportarían más información.

5.4.2.3. Comparativa entre d'Hondt y ruleta para $d = 50$

Al igual que para $d = 10$, se presentan a continuación la comparación de los estudios en relación al algoritmo con operador selección d'Hondt, escenarios 50.D.1000.0005 y 50.D.2000.0005 y con operador selección ruleta, escenario 50.R.1000.0005, para la complejidad $d = 10$, según los 5 criterios del PDEC-05 (apéndice D).

De entre los cuatro tamaños de población estudiados en el análisis del operador, se ha escogido el mejor que es tanto para el operador d'Hondt como para la ruleta: $n = 1000$ (escenarios 50.D.1000.0005 y 50.R.1000.0005 respectivamente). Se ha incluido además una comparación favorable a la ruleta, solo que esta vez, al revés que para $d = 10$, el tamaño de población no es el mismo: escenario 50.D.2000.0005 para el d'Hondt (el segundo tamaño de población estudiado en el análisis del operador) y 50.R.1000.0005 para la ruleta (su mejor resultado).

1. Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) y en la finalización ($5 \cdot 10^5$),

para cada ejecución (dada por Ter_Err o Max_FES).

Para la comparativa de los escenarios con el mismo tamaño de población para el d'Hondt y la ruleta, 50.D.1000.0005 y 50.R.1000.0005, partiendo de los resultados mostrados en la tabla 5.20 y en las gráficas 5.32 y 5.33, vemos como prácticamente siempre se muestra el d'Hondt superior a la ruleta, tanto para el error mínimo como para el error medio.

Si analizamos la comparación favorable a la ruleta (con el segundo tamaño de población estudiado con d'Hondt), 50.D.1000.0005 y 50.R.1000.0005, vemos en la tabla 5.20 y en las gráficas 5.34 y 5.35 que las superioridad ya no es tan amplia como en la comparación anterior. Si bien, aunque tras 5e5 FES, tanto para el error mínimo como para el error medio, el d'Hondt es mejor que la ruleta, no es hasta pasadas las 1e5 FES cuando esto ocurre. En FES menores o iguales a 1e5, la ruleta muestra un mejor resultado que el d'Hondt.

En toda la comparativa, el mejor resultado lo obtiene con el operador selección d'Hondt con un valor de error de 38,64.

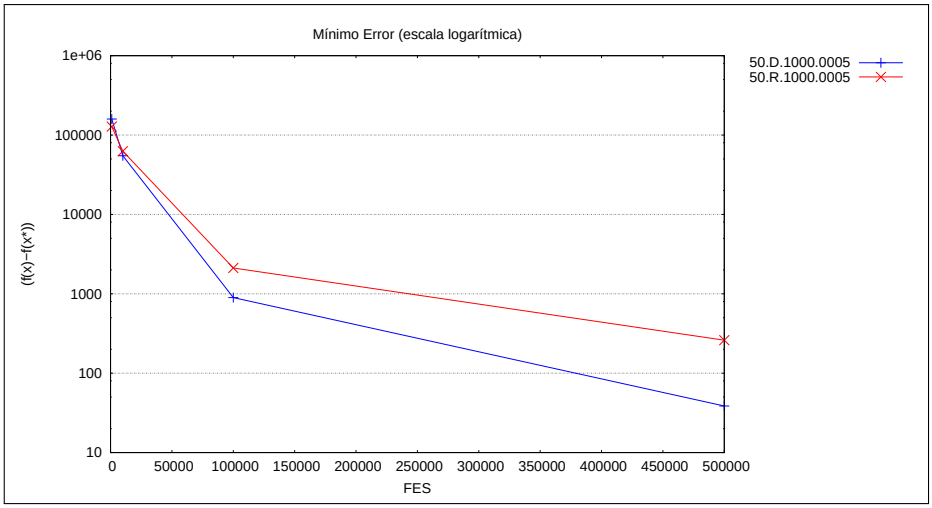

Figura 5.32: Mínimo error, según número de evaluaciones. Comparativa escenarios 50.D.1000.0005 y 50.R.1000.0005

FES \ Escenario		50.D.1000.0005	50.R.1000.0005	50.D.2000.0005
1e3	1st (Best)	159220,5	127885,79	
	7st	168188,05	166456,67	
	13st (Median)	175322,94	173320,02	
	19st	183275,55	179391,58	
	25st (Worst)	194692,33	196269,89	
	Mean	175535,24	170657,52	
	Std	9861,44	15526,14	
1e4	1st (Best)	54912,22	62594,12	89129,18
	7st	60542,13	77651,02	101062,09
	13st (Median)	65509,75	82893,69	104813,37
	19st	69691,22	86775,58	108585,08
	25st (Worst)	76333,75	97233,86	118493,3
	Mean	65597,07	81705,04	104935,94
	Std	5935,8	8955,93	7464,76
1e5	1st (Best)	896,09	2116,33	2523,23
	7st	1368,07	2417,13	2828,41
	13st (Median)	1506,62	2841,59	3175,88
	19st	1636,05	3185,13	3722,26
	25st (Worst)	2311,42	5076,79	4043,38
	Mean	1505,99	2950,49	3225,56
	Std	339,14	725,74	457,49
5e5 (end)	1st (Best)	38,64	260,46	39,82
	7st	114,02	496,18	79,81
	13st (Median)	199,18	562,3	114,61
	19st	258,57	705,4	128,49
	25st (Worst)	724,92	1896,08	518,87
	Mean	209,55	658,48	127,09
	Std	138,91	329,5	95,21

Tabla 5.20: Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) y en la finalización ($5 \cdot 10^5$), para cada ejecución. Comparativa escenarios 50.D.1000.0005, 50.R.1000.0005 y 50.D.2000.0005

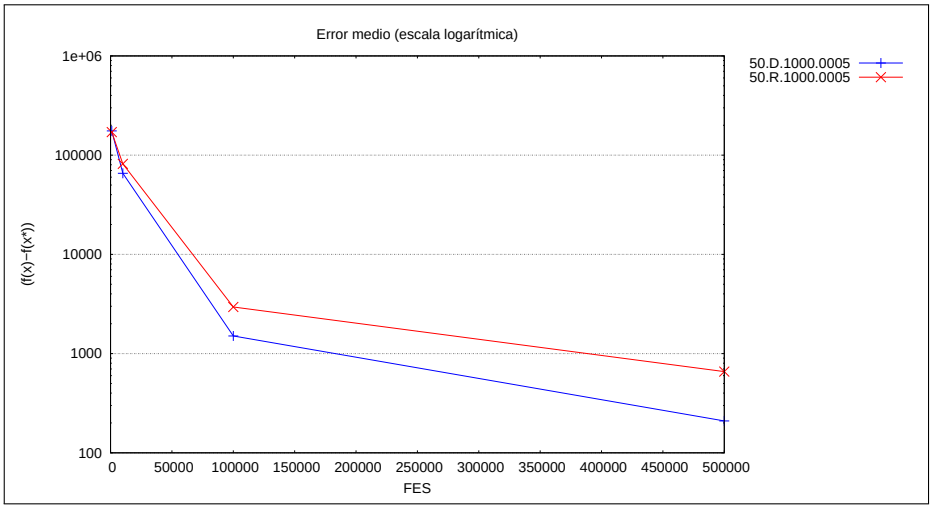

Figura 5.33: Error medio, según número de evaluaciones. Comparativa escenarios 50.D.1000.0005 y 50.R.2000.0005

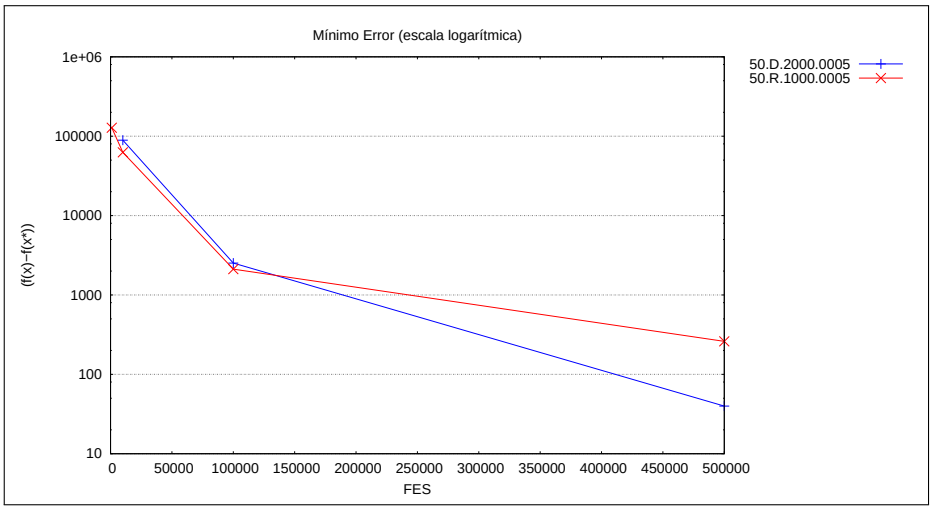

Figura 5.34: Mínimo error, según número de evaluaciones. Comparativa escenarios 50.D.2000.0005 y 50.R.1000.0005

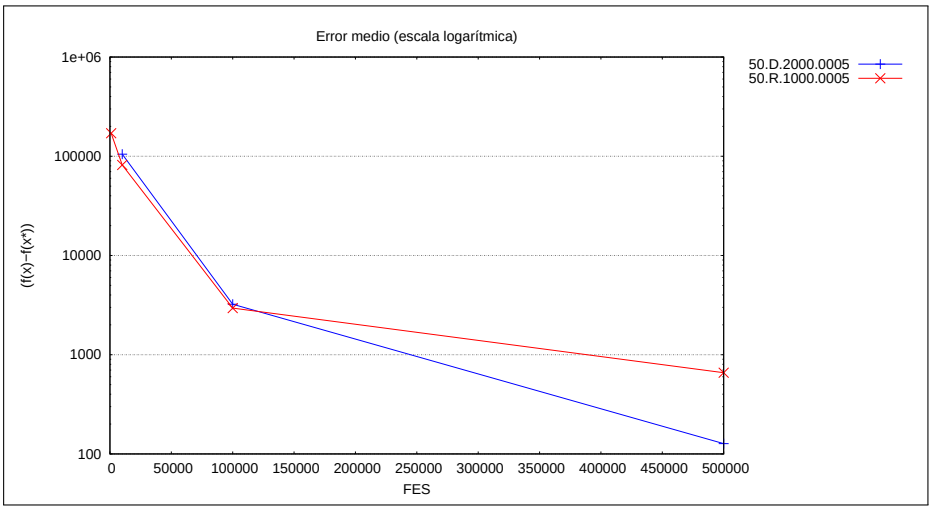

Figura 5.35: Error medio, según número de evaluaciones. Comparativa escenarios 50.D.2000.0005 y 50.R.2000.0005

2. Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el nivel de precisión. Ratio de éxito y rendimiento de éxito.

   Para el apartado de número de evaluaciones necesario para alcanzar una precisión determinada, se obtienen distintos resultados dependiendo del grado de precisión. Es importante aclarar que el valor MEDIO, ya que se computa sobre el número de veces que se alcanza el nivel de precisión puede llevar a error. Por ejemplo puede resultar que para un tamaño de población que solo alcance la precisión una o dos veces, pero con un número bajo de FES, contrasta con que para otro tamaño de población se alcance muchas más veces, con valores tan buenos como los del primero, pero que un par de malos resultados desvirtúen la media. Por ello solo vamos a analizar los FES alcanzados y será en los ratios, que contrastan FES y número de veces que se alcanza, los que nos aclararán la calidad del algoritmo.

   Como se puede observar en la tabla 5.21 y en las gráficas 5.36, 5.37, 5.38, 5.39, 5.40 y 5.41, en relación a la dimensión $d = 50$, el ratio de éxito para niveles de precisión medios y altos es 0, es decir que no se alcanzan tales valores de precisión. Tanto es así que se ha tenido que incluir incluso un valor de precisión muy bajo, 1e3.

   Para un nivel de precisión ya bajo, 1e2, solamente es alcanzado por los estudios con el d'Hondt, mientras que la ruleta no llega. Esto nos va

adelantando ya la superioridad del d'Hondt también para esta dimensión.

El resultado con la precisión muy baja, 1e3, muestra que el d'Hondt, tanto para la comparativa 50.D.1000.0005 y 50.R.1000.0005, como para 50.D.2000.0005 y 50.R.1000.0005 es mejor que para la ruleta. Así por ejemplo para ratio de rendimiento de éxito, tenemos que el escenario 50.D.1000.0005 obtiene 291871,96 FES frente a 291871,96 de la ruleta (50.R.1000.0005), que se ve empeorado al no alcanzar el valor 1 en el ratio de éxito; e incluso comparando con el escenario 50.D.2000.0005 (más favorable a la ruleta), tenemos 183280.

No tan aventajado para el d'Hondt es menor número de FES para la comparativa 50.D.2000.0005 y 50.R.1000.0005, ya que se obtienen valores muy similares: 162000 y 167000 respectivamente; aunque si utilizamos el mismo tamaño de población (50.D.1000.0005 y 50.R.1000.0005) la balanza se vuelve a inclinar del lado del d'Hondt con 91000 FES.

n	1st (Best)	7st	13st (Median)	19st	25st (Worst)	Mean	Std	Nivel de precisión	Success Rate	Success Perform.
50.D.1000.0005	-	-	-	-	-	-	-	1,00E-006	-	-
50.R.1000.0005	-	-	-	-	-	-	-	1,00E-006	-	-
50.D.2000.0005	-	-	-	-	-	-	-	1,00E-006	-	-
50.D.1000.0005	-	-	-	-	-	-	-	1,00E-001	-	-
50.R.1000.0005	-	-	-	-	-	-	-	1,00E-001	-	-
50.D.2000.0005	-	-	-	-	-	-	-	1,00E-001	-	-
50.D.1000.0005	-	-	-	-	-	-	-	1,00E+000	-	-
50.R.1000.0005	-	-	-	-	-	-	-	1,00E+000	-	-
50.D.2000.0005	-	-	-	-	-	-	-	1,00E+000	-	-
50.D.1000.0005	-	-	-	-	-	-	-	1,00E+001	-	-
50.R.1000.0005	-	-	-	-	-	-	-	1,00E+001	-	-
50.D.2000.0005	-	-	-	-	-	-	-	1,00E+001	-	-
50.D.1000.0005	279000	-	-	-	-	371000	82779,83	1,00E+002	0,2	1855000
50.R.1000.0005		-	-	-	-	-	-	1,00E+002	-	-
50.D.2000.0005	354000	454000	-	-	-	437555,56	40072,16	1,00E+002	0,36	1215432,1
50.D.1000.0005	91000	116000	130000	140000	228000	130360	26573,61	1,00E+003	1	130360
50.R.1000.0005	167000	218000	247000	304000	-	268521,74	80916,27	1,00E+003	0,92	291871,46
50.D.2000.0005	162000	170000	180000	190000	234000	183280	18183,14	1,00E+003	1	183280

Tabla 5.21: Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión, ratio de éxito y rendimiento de éxito. Comparativa escenarios 50.D.1000.0005, 50.R.1000.0005 y 50.D.2000.0005.

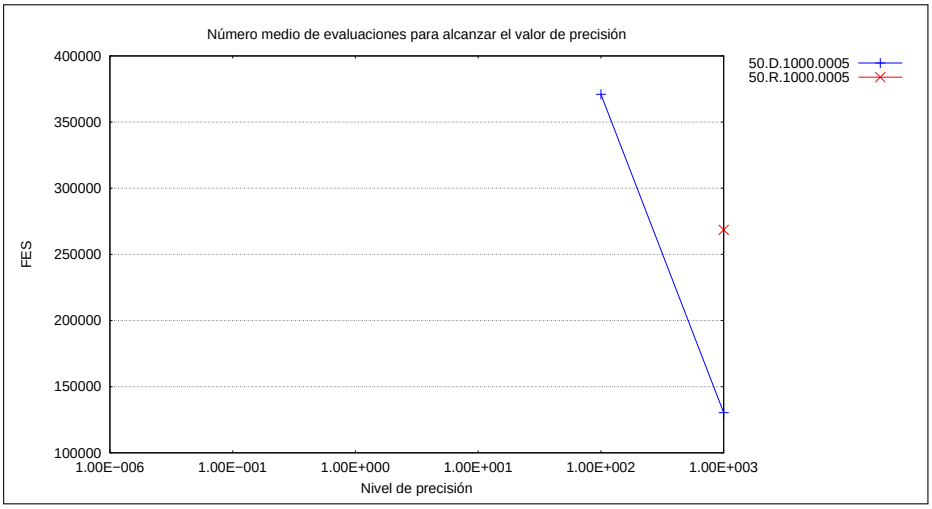

Figura 5.36: Número medio de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 50.D.1000.0005 y 50.R.1000.0005

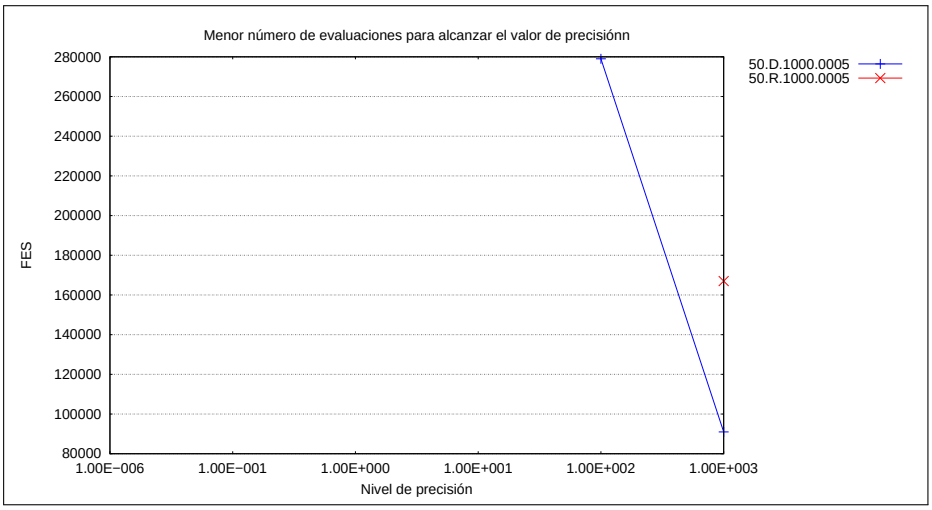

Figura 5.37: Menor número de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 50.D.1000.0005 y 50.R.1000.0005

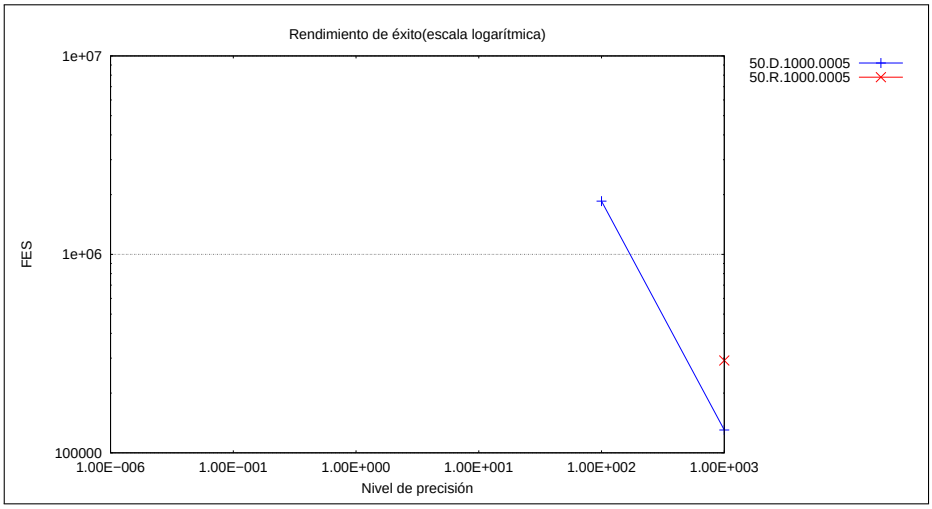

Figura 5.38: Rendimiento de éxito. Comparativa escenarios 50.D.1000.0005 y 50.R.1000.0005

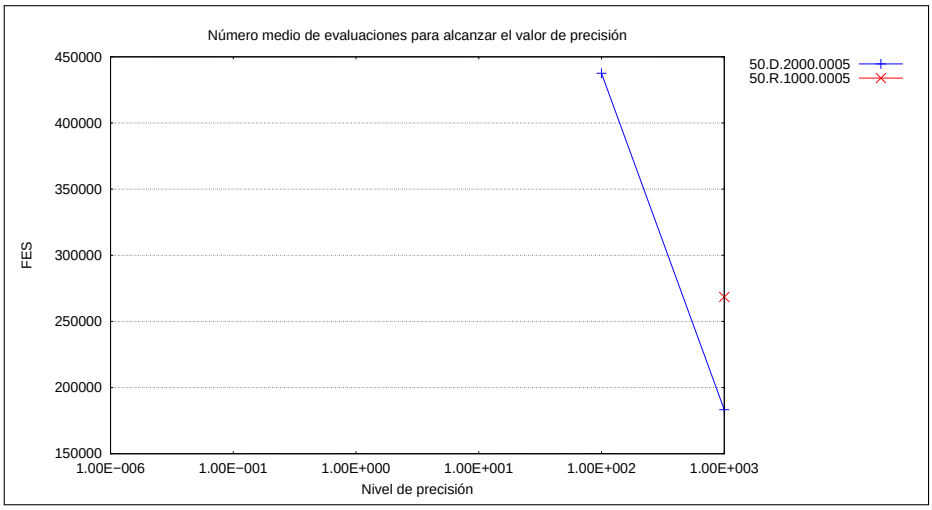

Figura 5.39: Número medio de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 50.D.2000.0005 y 50.R.1000.0005

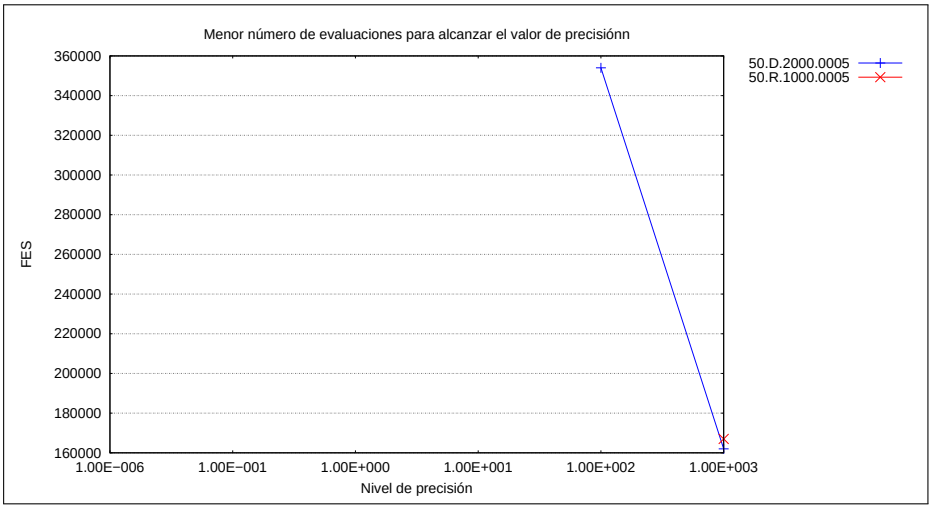

Figura 5.40: Menor número de evaluaciones para alcanzar el valor de precisión. Comparativa escenarios 50.D.2000.0005 y 50.R.1000.0005

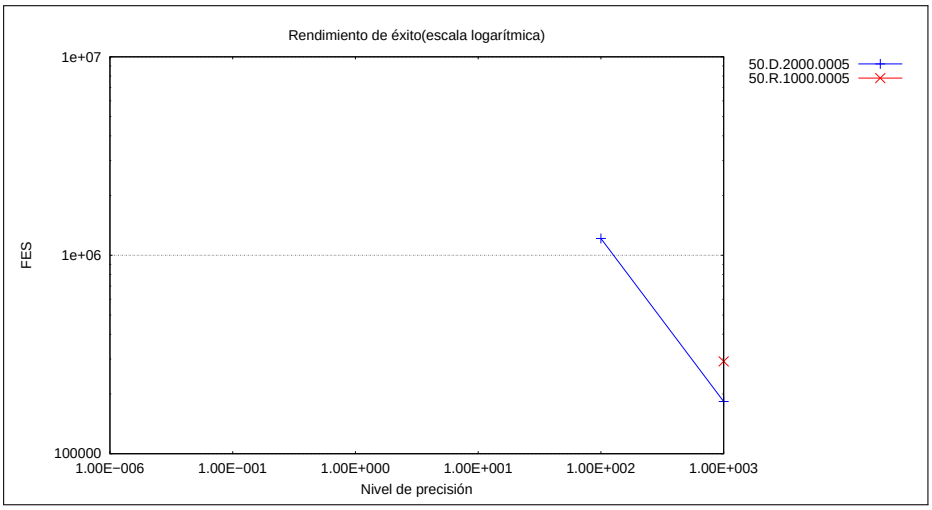

Figura 5.41: Rendimiento de éxito. Comparativa escenarios 50.D.2000.0005 y 50.R.1000.0005

3. Gráficas de convergencia.

   Observando las gráficas 5.42, 5.43 y 5.44, vemos como para 50.D.1000.0005 y 50.R.1000.0005, todas o casi todas las ejecuciones convergen ya casi al final, cerca de 5e5. No es así para el caso de 50.D.2000.0005, donde solo algunas ejecuciones logran converger al final.

   Como ya se comentó para $d = 10$ las gráficas más interesantes son 5.45 y 5.46, que muestran la comparativa de los escenarios 50.D.1000.0005 - 50.R.1000.0005 y 50.D.2000.0005 - 50.R.1000.0005 respectivamente. Así, para 50.D.1000.0005 y 50.R.1000.0005, vemos como claramente el d'Hondt supera a la ruleta y solo alguna de las mejores ejecuciones de la ruleta logran igualar a las peores del d'Hondt.

   Para 50.D.2000.0005 y 50.R.1000.0005, como ya se vio en el punto 1 (valor de error de la función), se observa muy claramente como en un rango de 0 a aproximadamente 200000 FES la ruleta obtiene menor error que el d'Hondt, pero es a partir de ahí cuando la cosa cambia y finalmente, para 5e5 FES el estudio supera a la ruleta. Incluso se ve claramente que si el límite de FES fuera mayor, 50.D.2000.0005 sería muy superior pues la tendencia que muestra es la de seguir mejorando ya que todavía no ha convergido.

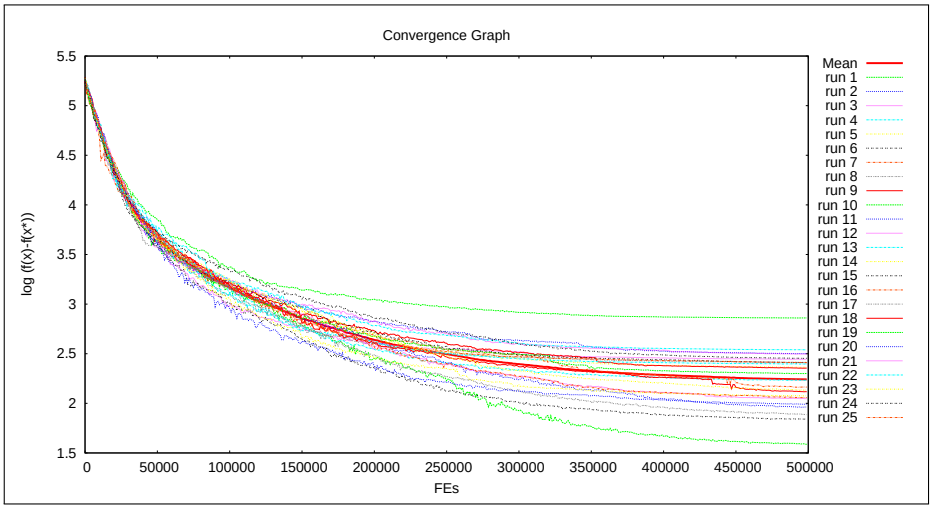

Figura 5.42: Convergencia. Escenario 50.D.1000.0005

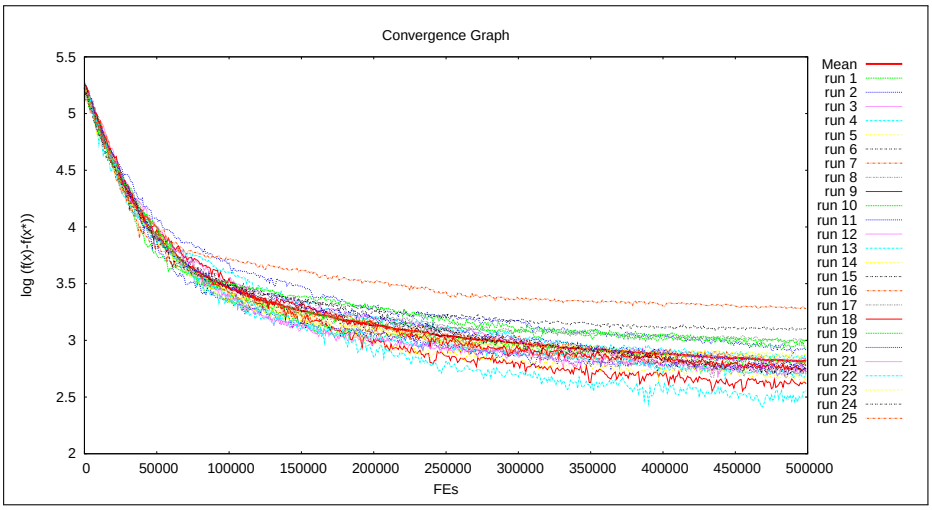

Figura 5.43: Convergencia. Escenario 50.R.1000.0005

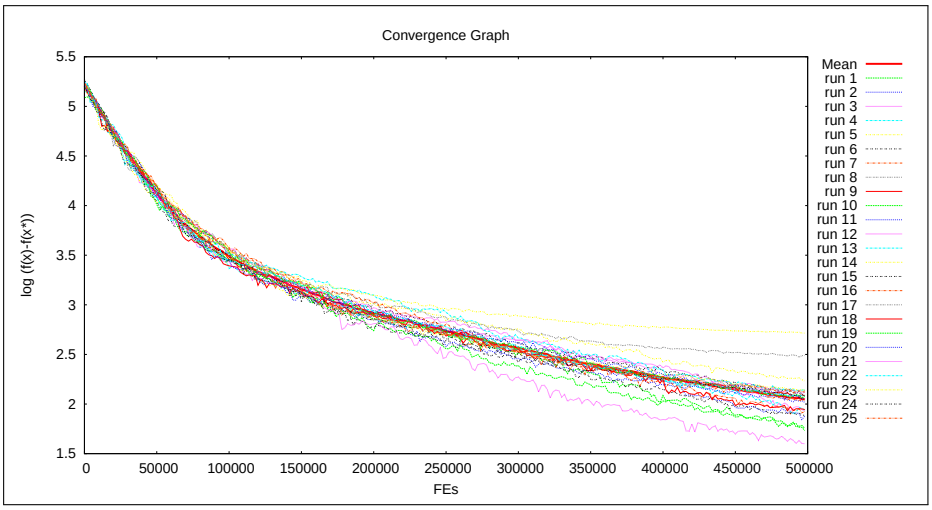

Figura 5.44: Convergencia. Escenario 50.D.2000.0005

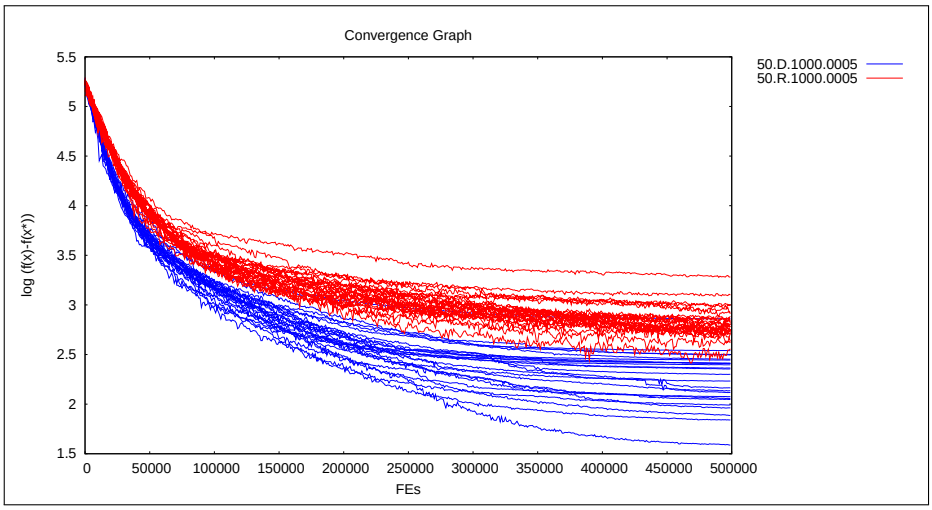

Figura 5.45: Convergencia. Comparativa escenarios 50.D.1000.0005 y 50.R.1000.0005

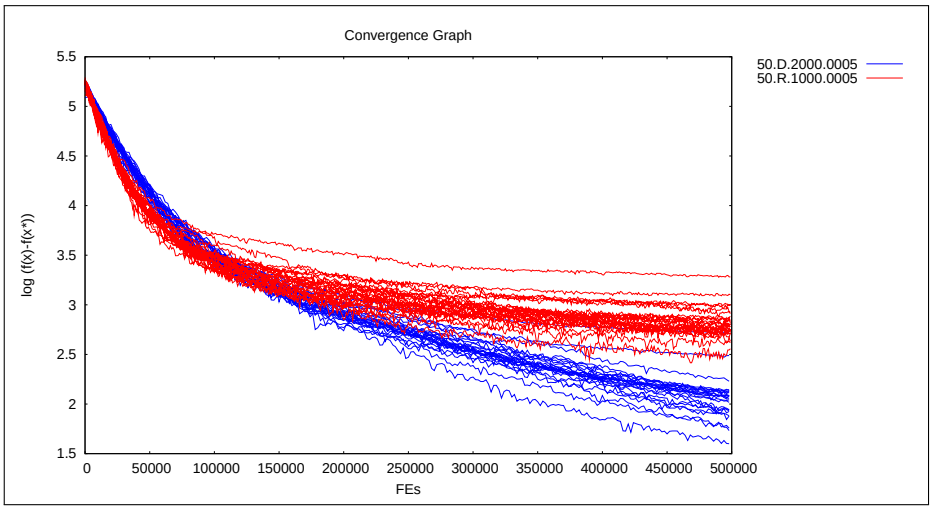

Figura 5.46: Convergencia. Comparativa escenarios 50.D.2000.0005 y 50.R.1000.0005

4. Algorithm Complexity.

   Tal como sucedía en $d = 10$, atendiendo a los valores de la tabla no podemos sacar ninguna conclusión.

Escenario	T0	T1	T2'	(T2'-T1)/T0
50.D.1000.0005	0,18	1,77	500,62	2771,4
50.R.1000.0005			498,67	2760,55
50.D.2000.0005			507,66	2810,48

Tabla 5.22: Algorithm Complexity- computing_time_t2. Comparativa escenarios 50.D.1000.0005, 50.R.1000.0005 y 50.D.2000.0005

5.4.3. Resultados media complejidad ($d = 30$)

Como se vio en el apartado 5.1.3 para la complejidad del algoritmo la estrategia consistía en realizar primeramente los estudios de la baja y la alta complejidad. Una vez finalizados dichos estudios, se ha visto que hay superioridad de los resultados para $d = 10$ sobre $d = 50$. Por lo tanto podemos suponer que esta superioridad también se va a dar sobre $d = 30$. Partiendo de esto podemos obviar realizar los estudios en la complejidad intermedia $d = 30$.

5.4.4. Resumen de resultados

A continuación se resume el análisis de los resultados obtenidos en este capítulo bajo la condiciones descritas: sobre el AGS con operador selección ley d'Hondt frente al AGS convencional (objetivo principal, sección 1.2), basados en los 5 criterios del PDEC-05 (apéndice D) utilizando la Función Esfera Desplazada (apartado 3.2.2).

- Resultados para baja complejidad ($d = 10$) Según se ha podido ver en el análisis, no cabe duda que para la complejidad $d = 10$, el operador selección ley d'Hondt obtiene resultados mucho mejores que la ruleta, tanto si se compara el mejor escenario de cada operador como si se escoge para los dos, el tamaño de población favorable a la ruleta.

  Así, por ejemplo, en lo que a error mínimo se refiere, la mejor ejecución que se obtiene en el estudio con ley d'Hondt es de 0,0008, muy por debajo de los 6,26 que alcanza la ruleta. Por otro lado, se ha visto que solamente el

estudio con la ley d'Hondt alcanza valores altos de precisión y con valores bajos se muestra muy superior a la ruleta en estos registros e incluso en los ratios de éxito y de rendimiento. Utilizando las gráficas de convergencia, se observa rotundamente la superioridad de las ejecuciones de la ley d'Hondt sobre la ruleta.

Por último, muestran complejidades similares, lo que indica que aunque elijamos el operador selección ley d'Hondt o la ruleta, la carga de uso de recursos del ordenador es casi igual.

- Resultados para baja complejidad ($d = 50$) Al igual que para $d = 10$, el operador selección d'Hondt se muestra superior a la ruleta, en cuanto a resultados del análisis se refiere.

  Buena prueba de ello son las gráficas de convergencia que comparan 50.D.1000.0005-50.R.1000.0005 y 50.D.2000.0005-50.R.1000.0005 (gráficas 5.45 y 5.46 respectivamente). Además la mejor ejecución que se obtiene con el d'Hondt es 38,64 frente a los 260,46 de la ruleta. En cuanto a los valores de precisión alcanzados, como ya se ha comentado en la sección 5.4, un nivel de precisión ya bajo, 1e2, solamente es alcanzado por los estudios con el d'Hondt, mientras que la ruleta no llega.

  Para finalizar, las complejidades son prácticamente iguales, aunque mucho mayores que $d = 10$ (lógicamente concuerda, al ser más iteraciones), esto es que aunque elijamos el operador selección d'Hondt o la ruleta, la carga de uso de recursos del ordenador es casi la misma.

Como cierre al capítulo, una vez terminada y analizada la experimentación, podemos decir que se ha cumplido el objetivo principal del proyecto, el cual, se recuerda que consiste en *comparar el comportamiento del algoritmo genético simple -con el operador Ruleta- en la resolución de un problema de optimización representativo, frente a una variante del algoritmo genético simple que incorpore un nuevo operador selección basado en alguna de las fórmulas electorales de divisores comunes.*

Capítulo 6

Conclusiones y trabajo futuro

Los capítulos precedentes son el resultado de posiblemente la primera exploración sobre el uso de fórmulas electorales en los algoritmos genéticos. Como se comentó en la introducción, este estudio se ha centrado en la aplicación de fórmulas de divisores comunes, en concreto la ley d'Hondt, como mecanismo de selección.

A continuación se comentan los aspectos más importantes de este proyecto y sus conclusiones. Además se proponen cuáles podrían ser y cómo se podrían orientar los trabajos futuros relacionados con el proyecto.

6.1. Conclusiones

En el desarrollo de este primer estudio, se ha partido de los objetivos a cumplir enunciados en la sección 1.2. Primeramente ha sido necesario recurrir a bibliografía sobre ciencia política y sistemas electorales. También se ha realizado la búsqueda tanto en Internet como en la bibliografía de baterías de funciones test y procedimientos de evaluación, así como una minuciosa búsqueda de aplicaciones informáticas de computación evolutiva que permitieran la ejecución de algoritmos genéticos.

Sobre el software de computación evolutiva escogido se ha implementado el nuevo operador selección además de otros ajustes. Adicionalmente, se ha creado un nuevo software para tratamiento de datos acorde a la test suite. Por último, se ha realizado la experimentación y se han analizado los resultados según el procedimiento de evaluación.

A continuación se presentan las conclusiones, en relación al objetivo principal, a

los objetivos instrumentales y al desarrollo general del proyecto.

6.1.1. Acerca del objetivo principal

Atendiendo a los objetivos enunciados en el primer capítulo, podemos decir que se ha cumplido el objetivo principal del proyecto: *comparar el comportamiento del algoritmo genético simple -con el operador Ruleta- en la resolución de un problema de optimización representativo, frente a una variante del algoritmo genético simple que incorpore un nuevo operador selección basado en alguna de las fórmulas electorales de divisores comunes.*

Tal como se citaba en la sección 1.2, el planteamiento genérico consistía en *explorar cómo funcionaría un algoritmo genético con un mecanismo de selección que utilice alguna de estas fórmulas frente a un algoritmo genético que utilice los operadores de selección habituales*. A la vista de los resultados del capítulo de experimentación (capítulo 5), como **conclusión principal** se puede asegurar que, para las condiciones experimentadas, el operador selección d'Hondt es superior a la ruleta. Bajo los criterios del procedimiento de evaluación escogido, obtiene valores que se acercan mucho más al óptimo, convergencias más tardías (lo que implica que tendría margen para seguir mejorando) y mejores registros. En ambos operadores la complejidad es similar, esto es, igual carga de recursos computacionales.

Gracias a los resultados altamente satisfactorios resulta evidente la necesidad de seguir experimentando este nuevo operador frente a otros escenarios, funciones, opciones, etc., con el fin de comprobar dónde se comporta mejor o peor, qué ventajas y desventajas ofrece su elección y sobre todo averiguar su potencial en la resolución de problemas de optimización con algoritmos genéticos.

6.1.2. Acerca de los objetivos instrumentales

Como se justificó en la sección 3.3, de entre todas las fórmulas electorales de divisores comunes, se escogió la ley d'Hondt, como el método de selección a probar en los algoritmos genéticos. Esta elección se corresponde con el objetivo instrumental 1 del proyecto. Es una de las fórmulas más populares en Europa y, en particular, la que se emplea en el sistema electoral español. Teniendo en cuenta este hecho, a la vista de los resultados y analizando la ley d'Hondt contra otras variantes, por ejemplo la Imperiali (más conservadora pero a priori similar), podemos decir que, entre las presentadas en el marco teórico, es una de las alternativas más interesantes para nuestros propósitos.

Con el propósito de cumplir con los objetivos instrumentales 2, 3 y 4, se ha realizado una minuciosa búsqueda de software de computación evolutiva, baterías de funciones test y procedimientos de evaluación. Con ello, se dispone ahora de una escogida colección a disposición para su uso en futuros estudios, como los especificados en el capítulo 3 y el apéndice D.

En relación al objetivo instrumental 4, para la selección de las alternativas de software de computación evolutiva evaluadas se recurrió a la decisión multicriterio. De entre estas alternativas, finalmente han resultado las más interesantes el software de algoritmos genéticos SGA-C -basado en el AGS (algoritmo genético simple, apartado 2.1.5) según lo enuncia Goldberg-, y GA toolbox (en C++). Como se vio en la sección 3.1 y se justificó en apéndice A, el software escogido para la experimentación fue GA toolbox. La decisión se fundamenta, por un lado, en estar desarrollado en el laboratorio Illigal [1], de reconocida trayectoria en este campo, y por otro, en la gran multitud de opciones que ofrece. Teniendo en cuenta lo anterior y dada la atractiva idea de seguir probando este tipo de fórmulas, podemos decir que la decisión de seleccionar GA toolbox ha permitido cumplir con los objetivos y facilita la continuación del trabajo realizado.

En cuanto a las funciones test y procedimientos de evaluación, inicialmente se pensó en contar con las funciones de De Jong (ver apéndice D), debido al reconocimiento con el que cuentan en la bibliografía. Tras un estudio en mayor profundidad, encontramos como alternativa el PDEC-05 (Problem Definitions and Evaluation Criteria for the CEC 2005 [SHL$^+$05]). Éste, a diferencia de las demás opciones, constituye una test suite como tal, al definir conjuntamente un grupo de funciones test y un completo procedimiento de evaluación. Anualmente, desde el año 2005 y acorde con las sucesivas ediciones del CEC (IEEE Conference on E-Commerce Technology), se proponen nuevas especificaciones de PEDEC con distintos objetivos (así por ejemplo, para el CEC06 se orientaron a problemas con restricciones). Se observa pues que hay una consolidación y aceptación del procedimiento, además de una importante difusión, habida cuenta del entorno donde se presenta. Los aspectos anteriores indican que el PDEC-05 cumple exitosamente con los objetivos instrumentales 2 y 3.

6.1.3. Conclusiones del desarrollo

Como se ha comentado, en los primeros pasos de este proyecto ha sido necesario recurrir a bibliografía sobre ciencia política y sistemas electorales. En particular

[1] Illigal (Illinois Genetic Algorithms Laboratory) es un laboratorio ubicado en la universidad de Illinois y dirigido por David E. Goldberg

se han consultado con mayor detalle los trabajos Cotarelo [CS92] y Vallés [CG97] (ver sección 2.2). Con ello, se ha conseguido una recopilación representativa de las fórmulas electorales que se utilizan en este campo.

Durante la ejecución de los objetivos fundamentales y el objetivo principal se realizaron además las siguientes tareas: programación del operador selección, adaptaciones sobre el software de algoritmos genéticos y la creación de un nuevo software para tratamiento de datos acorde a la test suite.

El proceso de ejecución del operador selección basado en la ley d'Hondt, implementado en GA toolbox, consiste principalmente en calcular los cocientes de los valores de salud de cada individuo (que equivalen al número de votos alcanzados) por los divisores (1, 2, 3, 4, etc.) y tomar como candidatos (a ser progenitor) los individuos correspondientes a los mayores valores de cociente (incluidos los de divisores 2, 3, 4, etc.). En un primer planteamiento se recurrió a una matriz de $n\ x\ n$ elementos ($n =$ nº de individuos), en la cual guarda los distintos valores resultantes del cociente de la **función de salud de cada individuo** por los divisores (1-2-3-4, etc.) y su ordenación posterior (de mayor a menor). Sin embargo, el cálculo y almacenamiento de esta matriz en cada iteración, consume muchos recursos informáticos, lo que hace que para altas poblaciones resultase un algoritmo muy lento (por ejemplo, para una población de 200 individuos, serían 40000 elementos). Como alternativa, se recurrió a utilizar un vector de una sola dimensión, que se inicializa con los valores de salud de los respectivos individuos. En este vector, se van seleccionando individuos de uno en uno y se cambian los valores de salud respectivos por el cociente del número de veces que haya sido seleccionado. Así, el enfoque del uso en un vector ha sido mucho más acertado que con el enfoque inicial sobre una matriz, ya que siendo el resultado el mismo, se requieren muchos menos recursos, por lo que el algoritmo no alarga sus tiempos de ejecución.

En general, salvo el problema planteado, en la programación del nuevo operador selección en GA toolbox no han surgido grandes complicaciones. Tras implementar este primer operador, resulta mucho más sencillo programar cualquier otro operador basado en fórmulas electorales de divisores comunes, debido a la similitud que presentan.

Tal como se ha visto en la sección 3.1 (tabla 3.1), hacía falta realizar una serie de modificaciones al GA toolbox para que funcionase como AGS. Las principales eran adaptar los operadores cruce y mutación simple y hacer una discretización inicial. La nueva versión de este software junto con el resto de modificaciones comentadas en la sección 4.2, recibe el nombre de MGA toolbox (Modified

Single and Multiobjective Genetic Algorithm Toolbox in C++). Cabe destacar que la adaptación llevada a cabo no resultó inmediata y exigió incurrir en un esfuerzo importante de desarrollo. A pesar de ello, las características de la versión resultante y la multitud de opciones del software, siguen respaldando la elección de este software.

En el desarrollo de GA toolbox (capítulo 4) se analizó cómo enfocar los resultados de las ejecuciones respecto a los 5 criterios del PDEC-05 para el procedimiento de evaluación (apéndice D). En primer lugar, para la recopilación de datos, se pensó en insertar diversas funciones que guardaran estos datos, pero pronto se vio que el código podía quedar muy alterado. Por ello se creó un nuevo programa para recopilación y tratamiento de datos acorde a los 5 criterios: Statistics. Según se indica en la sección 4.3, Statistics recoge los resultados volcados a fichero por el algoritmo genético y los procesa para mostrarlos de acuerdo con el formato de la test suite PDEC-05. El objetivo era crear un programa eficiente, portable y que permitiera continuar utilizando el mismo lenguaje de programación que GA toolbox, el lenguaje C++. Una vez se ha finalizado la experimentación hemos podido comprobar que la programación y uso del programa Statistics ha permitido aligerar fuertemente la recopilación de datos en la experimentación, ahorrando mucho tiempo. Permite además su aprovechamiento en sucesivos análisis.

El software al que se ha recurrido para la representación, a partir de la salida de Statistics, es GNUplot, un programa muy flexible para generar gráficas de funciones y datos, capaz de mostrar sus resultados directamente en pantalla, así como en multitud de formatos de imagen. GNUplot fue un candidato inmediato, al trabajar en Linux. No obstante, para su elección se tuvieron en cuenta otros factores como la ingente cantidad de material de ayuda disponible en Internet y el hecho de ser compatible con los sistemas operativos más populares. Durante toda la experimentación, GNUplot ha funcionado perfectamente ofreciendo las gráficas deseadas. El único punto que ha conllevado cierta dificultad ha sido adaptar los scripts (fichero de instrucciones informáticas) a las necesidades específicas de nuestro proyecto, aunque rápidamente se ha podido manejar con soltura este aspecto, por lo que vemos que es una decisión adecuada para nuestro propósito.

El desarrollo se completó con una serie de operaciones relativas al uso del PDEC-05. La más importante fue la implementación de las funciones del PDEC-05 en el GA toolbox, que se realizó de un modo sencillo, apoyando su selección como test suite para nuestro proyecto.

En definitiva, una vez analizadas las soluciones adoptadas y el trabajo realizado, podemos concluir que globalmente este proyecto, además de arrojar unos resultados muy prometedores, sienta las bases y deja orientado el trabajo para continuar avanzando en la línea marcada por el propósito inicial. A este respecto, se presenta en el siguiente apartado una propuesta de trabajos futuros.

6.2. Trabajos futuros

Según se ha remarcado al inicio, el planteamiento genérico del proyecto formulado en la introducción consiste en explorar cómo funcionaría un algoritmo genético con un mecanismo de selección que utilice alguna de las fórmulas electorales de divisores comunes frente a un algoritmo genético que utilice los operadores de selección habituales. Como se indicó en el planteamiento de objetivos (ver 1.2), para poder llevarlo a cabo había que recurrir a una comparación del comportamiento entre diversos algoritmos genéticos con diferentes operadores, fórmulas electorales y variantes de algoritmos genéticos. Debido al alcance del propio proyecto, este extenso planteamiento implicó acotar las dimensiones de la comparación a un conjunto representativo. Se acaba de ver en el apartado de conclusiones, que el proyecto se ha dejado orientado a continuar con la exploración. Es decir, a poder seguir desarrollando el planteamiento genérico, con nuevos operadores, variantes, funciones objetivo, etc.

Caso de resultar positivo el resultado del análisis del planteamiento genérico al completo, cabría esperar que el empleo de las fórmulas electorales en los algoritmos genéticos se extendiera dentro del mundo de la computación evolutiva. En particular, estos nuevos operadores podrían estar presentes como opción disponible de operador selección en los software de algoritmos genéticos de uso más extendido o incluirse como unos operadores de selección más en la bibliografía especializada. Lógicamente, para poder llegar a este punto, queda abierto un amplio campo de investigación. Se comentan a continuación los caminos que se encuentran más interesantes para continuar el desarrollo del planteamiento genérico.

El camino más inmediato a seguir es explorar el funcionamiento del nuevo operador frente a otras funciones. En este proyecto se ha limitado el análisis a la función 1, función Esfera desplazada (De Jong 1), del conjunto de 25 funciones test del PDEC-05. Como ya se vio en el apartado 3.2.1, el conjunto cubre ampliamente cualquier tipo de problema de optimización. Con ello, quedan todavía 24 funciones sobre las que seguir explorando y de donde se pueden sacar muchas conclusiones del comportamiento.

Durante la realización de este proyecto, se obtuvo adicionalmente un subconjunto de 10 funciones de las 25 del PDEC, mostrado en el apéndice E, bajo la premisa de que este subconjunto fuera lo más completo y homogéneo posible. El objetivo es identificar, de entre el total de funciones pendientes, el subgrupo a considerar para continuar el estudio con garantías.

En segundo lugar, resultaría necesario conocer el comportamiento del nuevo operador frente a otros operadores selección distintos a la ruleta. Los demás operadores selección que ofrece GA toolbox son los siguientes: torneo con reemplazamiento, torneo sin reemplazamiento, truncado, (ruleta) y stochastic universal selection (SUS). Los tres primeros son considerados por Goldberg como operadores "avanzados" [Gol89a], mientras que el último podríamos denominarlo de última generación, lo que aporta gran interés a una comparación con la ley d'Hondt. Por último, un operador a tener en cuenta, no incluido en el software de genéticos, sería el del valor esperado ya que coincide con la variante Hare (fórmula electoral), que se programaría del mismo modo que se ha procedido con la ley d'Hondt.

Una vez explorados el comportamiento con otras funciones y operadores, el campo final estaría ya fuera del AGS. Se podrían entonces estudiar otros operadores cruce y mutación, añadir operadores nuevos, probar nuevas opciones, etc. Como se puede ver la exploración se podría ampliar a infinidad de alternativas. Para evitarlo habría que definir las comparativas que realmente fueran relevantes para cumplir con el planteamiento genérico.

Adicionalmente a la ley d'Hondt, se abre un amplio abanico muy prometedor de nuevas opciones de operador selección basadas en el resto de fórmulas electorales [CS92], en especial las fórmulas proporcionales de divisores comunes (de las que forma parte la ley d'Hondt, ver apartado 2.2.2.2). Estas fórmulas, apenas se han tenido en cuenta (al menos como tales funciones electorales), como técnicas de reparto en los métodos de resolución de problemas de optimización.

Como cierre a este apartado, cabe mencionar que un atractivo reto de este uso de las fórmulas, sería diseñar un algoritmo y probarlo en alguna de las ediciones o variantes del PDEC convocadas cada año en el CEC o en alguna competición de algoritmos evolutivos similar.

Referencias bibliográficas

[Adr88] Francisco Rodríguez Adrados. *La democracia ateniense*. Alianza, 1988.

[Bak85] J. E. Baker. *Adaptive selection methods for genetic algorithms*. Proceedings of the International Conference on Genetic Algorithms and Their Applications, páginas 101–111, 1985.

[Bak87] J. E. Baker. *Reducing bias and ineficiency in the selection algorithm*. Proceedings of the Second International Conference on Genetic Algorithms, 1987.

[BRP97] Sergio Barba-Romero y Jean-Charles Pomerol. *Decisiones multicriterio: Fundamentos teóricos y utilización práctica*. Servicio de Publicaciones de la Universidad de Alcalá, 1997.

[CG97] Josep M. Vallès Casadevall y Agustí Bosch Guardella. *Sistemas electorales y gobierno representativo*, capítulo 3. Ariel ciencia política, 1997.

[Coe95] Carlos A. Coello Coello. *Introducción a los Algoritmos Genéticos*. `http://www.redcientifica.com/doc/doc199904260011.html`, enero 1995.

[CS92] Ramón García Cotarelo y Juan Luis Paniagua Soto. *Introducción a la ciencia política*, capítulo 3. Universidad Nacional de Educación a Distancia, 1992.

[CW95] Arthur L. Corcoran y Roger L. Wainwright. *Using LibGA to Develop Genetic Algorithms for Solving Combinatorial Optimization Problems*. Reporte técnico, Department of Mathematical and Computer Sciences, the University of Tulsa, 1995.

[DA95] K. Deb y R Agarwal. *Simulated binary crossover for continuous search space*. Complex Systems, volumen 9, páginas 115–148, 1995.

[DB99] K. Deb y H. Beyer. *Self-Adaptive Genetic Algorithms with Sumulated Binary Crossover*. Reporte técnico, Department of Computer Science/XI, University of Dortmund, 1999.

[Deb01] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: John Wiley and Sons, 2001.

[DK95] K. Deb y A. Kumar. *Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems*. Complex Systems, volumen 9, páginas 431–454, 1995.

[For85] S. Forrest. *Documentation for prisoners dilemma and norms programs that use genetic algorithms*. Sin publicar, 1985.

[GB89] J. J. Grefenstette y J. E. Baker. *How genetic algorithms work: A critical look at implicit parallelism*. Proceedings of the Third International Conference on Genetic Algorithms, páginas 20–27, 1989.

[GNU09] *GNUplot*. `http://es.wikipedia.org/wiki/Gnuplot`, septiembre 2009.

[Gol89a] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley Co., 1989.

[Gol89b] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*, páginas 106–120. Addison-Wesley Co., 1989.

[GR87] D. E. Goldberg y J. J. Richardson. *Genetic algorithms with sharing for multimodal funcion optimization*. Proceedings of the Second International Conference on Genetic Algorithms, páginas 41–49, 1987.

[Har95] G. R. Harik. *Finding multimodal solutions using restricted tournament selection*. Proceedings of the Sixth International Conference on Genetic Algorithms, páginas 24–31, 1995. (IlliGAL Report No. 94002).

[Hol75] John Henry Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[JJLSQ06] S. Baskar J. J. Liang, P. Suganthan y A. K. Qin. *Performance Evaluation of Multiagent Genetic Algorithm*, volumen 5, número 1, páginas 83–96. Springer Netherlands, 2006.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[KPAM02] K.Deb, A. Pratap, S. Agrawak y T. Meyarivan. *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II*, volumen 6 (2), páginas 182–197. IEEE Transsactions on Evolucionay Computation, 2002.

[Mah92] S. W. Mahfoud. *Crowding and preselection revisited*. Parallel Problem Solving from Nature, volumen 2, páginas 27–36, 1992. (IlliGAL Report No. 92004).

[MF00] Zbigniew Michalewicz y David B. Fogel. *How to Solve It: Modern Heuristics*, capítulo 7. Springer-Verlag, 2000.

[Mic96] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, capítulo 1-4 (Part I). Springer-Verlag, 3ª edición, 1996.

[MSV93] H. Mühlenbein y D. Schlierkamp-Voosen. *Predictive models for the breeder genetic algorithm: I. continous parameter optimization*. Evolutionary Computation, volumen 1, páginas 25–49, 1993.

[Nes07] Sergio Nesmachnow. *Algoritmos Evolutivos*. `www.fing.edu.uy/inco/cursos/geneticos/ae/2009/Clases/clase3.pdf`, 2007.

[NM65] J. A. Nelder y R. Mead. *A simplex method for function minimization*. The Computer Journal, volumen 8, páginas 308–313, 1965.

[Par08] Abelardo Pardo. *Programación en ensamblador de la arquitectura IA-32. OpenCourseWare*. `http://ocw.uc3m.es/ingenieria-telematica/arquitectura-de-ordenadores/lecturas/html/arc.html`, junio 2008.

[Pas91] Manuel Pastor. *Ciencia política*, capítulo 7. McGraw-Hill, 1991.

[PFTV89] W. Press, B. Flannery, S. Teukolsky y W. Vettering. *Numerical recipes in C*. Cambridge University Press, 1989.

[Poh94] Hartmut Pohlheim. *Genetic Algorithm Toolbox, Test Functions*. Reporte técnico, Department of Automatic Control and Systems Engineering, University of Sheffield, 1994.

[Poh06] Hartmut Pohlheim. *GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB Documentation*. `http://www.geatbx.com/docu/fcnindex-01.html`, diciembre 2006.

[Qua98] D. Quagliarella. *Genetic algorithms and evolution strategy in engineering and computer science: recent advances and industrial applications*. John Wiley & Sons, 1998.

[Rae71] Douglas W. Rae. *The Political Consequences of Electoral Laws*. Yale University Press, 1971.

[Rom93] Carlos Romero. *Teoría de la decisión multicriterio, conceptos, técnicas y aplicaciones*. Alianza, 1993.

[RP02] Piedad Tolmos Rodríguez-Piñero. *Introducción a los algoritmos genéticos y sus aplicaciones*. ASEPUMA X, 2002.

[Sas07] K. Sastry. *Single and Multiobjective Genetic Algorithm Toolbox in C++*. Reporte técnico, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign, junio 2007.

[Ser96] Anselmo Pérez Serrada. *Una introducción a la Computación Evolutiva*. Tesis, Universidad de Valladolid, 1996.

[SG01] K. Sastry y D. E. Goldberg. *Modeling tournament selection with replacement using apparent added noise*. Intelligent Engineering Systems Through Artificial Neural Networks, volumen 11, páginas 129–134, 2001.

[SGE94] Robert E. Smith, David E. Goldberg y Jeff A. Earickson. *SGA-C: A C-language Implementation of a Simple Genetic Algorithm*. Reporte técnico, The Clearinghouse for Genetic Algorithms, the University of Alabama, Department of Engineering Mechanics, marzo 1994.

[SGK99] K. Sastry, D. E. Goldberg y G. Kendall. *Genetic algorithms: Introductory Tutorials in Optimization, Search, and Decision Support Methodologies*. Reporte técnico, Berlin: Springer, 1999.

[SHL+05] P. Suganthan, N. Hansen, J. J. Liang1, K. Deb, Y. P. Chen, A. Auger y S. Tiwari. *Problem Definitions and Evaluation Criteria for the CEC 2005. Special Session on Real-Parameter Optimization*. Reporte técnico, School of Electrical and Electronic Engineering Nanyang Technological University, Singapore, 2005.

[SK06] P. Suganthan y Qin Kai. *Recent Advances in Real-Parameter Evolutionary Algorithms*. Reporte técnico, School of Electrical and Electronic Engineering Nanyang Technological University, Singapore, 2006.

[Whi89] D Whitley. *The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J.D. Schaffer (ed.)*. Proceedings of the Third Conference on Genetic Algorithms. Morgan Kauffmann Publisher, páginas 116–121, 1989.

[YBS+03] E. Yolis, P. Britos, J. Sicre, A. Servetto, R. García-Martínez y Perichinsky. G. *Algoritmos Genéticos aplicados a la categorización automática de documentos*. VIII Congreso Argentino de Ciencias de la Computación, 2003.

Apéndice A

Métodos multicriterio aplicados

En el apartado 3.1.2 se ha indicado que para la selección del software de computación evolutiva, se usaron 3 métodos multicriterio: sumas ponderas, TOPSIS y ELECTRE, además de mostrar los resultados de la aplicación de estos.

Este apéndice no pretende ser una introducción a los métodos multicriterio ni detallar sus fundamentos (para ello se recomienda consultar el [BRP97] de la bibliografía), sino que tiene como objetivo mostrar cómo se han aplicado estos métodos a la selección, incluyendo algunas aclaraciones teóricas.

A.1. Matriz de decisión y ponderación

Para poder aplicar la decisión multicriterio es necesario evaluar cuantitativamente los atributos de las alternativas. Esto se realiza asignando puntuaciones, que no es otra cosa que tomar valores para cada criterio. Una forma sencilla de representarlo, es recoger estos datos en una matriz (matriz de decisión) en la que las columnas expresan los criterios y las filas las alternativas a considerar. Así, el elemento a_{ij} indica el nivel que alcanza la alternativa A_i con respecto al criterio C_j

Por otro lado, la dificultad de la decisión multicriterio es elegir una alternativa globalmente preferida, cuando las opiniones del decisor sobre cada alternativa difieren para cada criterio. Por ello resulta necesario que el decisor exprese de forma cuantitativa la importancia que concede a cada criterio en esa decisión, en otras palabras, asignar un "peso"(W_j) a cada criterio acorde a su importancia

En la tabla A.1 se muestran las puntuaciones asignadas a cada alternativa para cada criterio[1] y los correspondientes pesos o ponderaciones de los criterios,

[1]Criterios: configurable (C_1), adaptabilidad a AGS (C_2), interface y definición de parámetros

partiendo de las observaciones recogidas en la tabla 3.1 *comparativa de las aplicaciones informáticas* de la sección 3.1.

Criterios / Alternativas	C1	C2	C3	C4	C5
SGA-C	0	5	3	3	5
GA toolbox	1	4	4	4	5
LibGA	1	3	2	2	2
SES	0	1	4	1	1
VectorGA	0	2	3	4	3
Pesos	2	7	2	6	3

Tabla A.1: Matriz de decisión y pesos

Para asignar las puntuaciones de los criterios, se ha seguido un conocido sistema, muy usado en encuestas y formularios, el cual consiste en asignar puntos según una escala en acorde al nivel de satisfacción de cada criterio. Para nuestro caso se ha establecido la siguiente escala: 5 (excelente), 4 (bien), 3 (aceptable), 2 (mal), 1 (muy mal), 0 (no corresponde, no disponible). En caso de que la valoración sea binaria (si o no) sería: 1 (si), 0 (no).

El método seguido para la asignación de los pesos a los criterios de selección es el método de las Puntuaciones, el cual consiste en pedir al decisor que reparta para nuestro caso 20 puntos, entre los distintos criterios que habremos de emplear posteriormente para evaluar a los proveedores. Se escogió este método debido a su sencillez y a su aplicación para este caso.

A.2. Normalización de puntuaciones y pesos

Otro de los problemas a los que se enfrenta la decisión multicriterio es la necesidad de comparar atributos diferentes y expresados en unidades diferentes, por lo que antes de comenzar a aplicar cualquier método multicriterio suele ser aconsejable, y en muchos casos necesario, normalizar las puntuaciones. Así pues, el objetivo de la normalización es obtener escalas comparables entre atributos, de forma que cada valor pueda establecerse como una unidad dimensional.

(C_3), salida y representación (C_4) y fuente (C_5). Consultar sección 3.1.

En ocasiones, cuando se asignan las puntuaciones en la matriz de decisión, los peores atributos de cada alternativa corresponden a los valores más altos y los mejores a los más bajos. En este caso, antes de proceder a la normalización es necesario que todos los criterios están MAXIMIZADOS, es decir que para todos los criterios se corresponda que los mejores atributos sean los que posean mayores valores y los peores los valores más bajos. Entonces, bastaría con utilizar la transformación $a'_{ij} = 1/a_{ij}$ en todos los elementos del criterio minimizado para pasar a la maximización. Para nuestro caso, todos los criterios están maximizados, por lo que no hay que realizar ninguna operación.

El método para la normalización de la matriz de decisión y para la ponderación que se ha escogido ha sido el procedimiento número 3 de entre los cuatro citados en la bibliografía [BRP97]. Se ha tomado este método por su interpretación sencilla (% del total $\sum_i a_i$) y porque conserva la proporcionalidad y favorece la concentración de valores (lo que nos ayudará en el método ELECTRE). Además, es el utilizado para la normalización de los pesos.

Este procedimiento consiste en normalizar cada columna de la matriz de decisión usando la expresión A.2, que no es otra cosa que para cada columna, dividir a cada elemento por la suma de todos los elementos de su misma columna.

$$r_{ij} = \frac{a_{ij}}{\sum_{i=1}^{m} a_{ij}}, \qquad (i = 1, \ldots, m; \quad j = 1, \ldots, n) \tag{A.1}$$

De manera análoga, para la ponderación se divide a cada peso por la suma de todos.

La tabla A.2 muestra las puntuaciones y los pesos en la matriz de decisión obtenidos después de la normalización.

r_{ij}	C1	C2	C3	C4	C5
A1	0	0,33	0,19	0,21	0,31
A2	0,5	0,27	0,25	0,29	0,31
A3	0,5	0,2	0,13	0,14	0,13
A4	0	0,07	0,25	0,07	0,06
A5	0	0,13	0,19	0,29	0,19
Pesos	0,1	0,35	0,1	0,3	0,15

Tabla A.2: Preparación de los datos - Matriz de decisión normalizada

A.3. Métodos aplicados

Una vez que se han concluido los pasos previos (asignación de puntuaciones y pesos y la preparación de los datos) estamos en condiciones de emplear los métodos multicriterio a la selección de la aplicación informática indicados en el apartado 3.1.2, en donde además se muestran sus resultados.

El motivo por el que se han elegido estos 3 métodos es primeramente por contraste entre varios métodos y en segundo lugar porque estos 3 métodos son los más comunes.

A.3.1. Sumas ponderadas

El método de las sumas ponderadas calcula un escalar para cada alternativa, partiendo de la matriz de decisión normalizada, según la siguiente expresión sumatoria:

$$V(A_i) = \sum_{j=1}^{n} W_j \cdot r_{ij}, \qquad (i = 1, \ldots, m) \tag{A.2}$$

Quedando para nuestro problema el siguiente resultado:

$W_j \cdot r_{ij}$	C1	C2	C3	C4	C5	$V(A_i)$
A1	0,000	0,333	0,188	0,214	0,313	0,247
A2	0,500	0,267	0,250	0,286	0,313	0,301
A3	0,500	0,200	0,125	0,143	0,125	0,194
A4	0,000	0,067	0,250	0,071	0,063	0,079
A5	0,000	0,133	0,188	0,286	0,188	0,179

Tabla A.3: Sumas ponderadas - matriz escalada

La alternativa preferida será la que mayor $V(A_i)$ obtenga. En este caso es la alternativa 2.

A.3.2. TOPSIS

A continuación aplicaremos el método TOPSIS a nuestro problema de decisión.

Partiendo de la matriz de decisión normalizada A.2, se calcula según la siguiente expresión la matriz de valores normalizados ponderados.

$$V_{ij} = W_j \cdot r_{ij}, \qquad (i = 1, \ldots, m;\ j = 1, \ldots, n) \tag{A.3}$$

$V_{ij} = W_j \cdot r_{ij}$	C1	C2	C3	C4	C5
A1	0,000	0,117	0,019	0,064	0,047
A2	0,050	0,093	0,025	0,086	0,047
A3	0,050	0,070	0,013	0,043	0,019
A4	0,000	0,023	0,025	0,021	0,009
A5	0,000	0,047	0,019	0,086	0,028

Tabla A.4: TOPSIS - matriz de valores normalizados ponderados

Con esta matriz ya se pueden identificar las soluciones ideal positiva y negativa en términos de los valores ponderados normalizados, según la siguiente expresión:

$$A^+ = (V_1^+ \cdots V_n^+) = (Max_0(V_{i0}), Max_1(V_{i1}) \cdots, Max_n(V_{in})), \quad (i = 1, \ldots, m) \tag{A.4}$$

$$A^+ = (V_1^- \cdots V_n^-) = (Min_0(V_{i0}), Min_1(V_{i1}) \cdots, Min_n(V_{in})), \quad (i = 1, \ldots, m) \tag{A.5}$$

Quedando para nuestro caso del siguiente modo:

						Solución ideal
A+	0,050	0,117	0,025	0,086	0,047	Más positiva
A-	0,000	0,023	0,013	0,021	0,009	Más negativa

Tabla A.5: TOPSIS - solución ideal más positiva y más negativa

Una vez obtenidas la solución ideal más positiva y la más negativa, se calculan las medias de separación con respecto a estas con las siguientes expresiones:

$$S_i^+ = \sqrt{\sum_{j=1}^{n} (V_{ij} - V_j^+)^2}, \qquad (i = 1, \ldots, m) \tag{A.6}$$

$$S_i^- = \sqrt{\sum_{j=1}^{n} (V_{ij} - V_j^-)^2}, \qquad (i = 1, \ldots, m) \tag{A.7}$$

Obtenemos así las siguientes distancias con respecto a la solución más positiva y la más negativa de cada alternativa.

S(A1)+	0,055
S(A1)-	0,110
S(A2)+	0,023
S(A2)-	0,114
S(A3)+	0,070
S(A3)-	0,072
S(A4)+	0,129
S(A4)-	0,013
S(A5)+	0,088
S(A5)-	0,071

Tabla A.6: TOPSIS - medias de separación

Por último calcularemos las similitudes a la solución ideal positiva (basándonos en las distancias), con la siguiente expresión:

$$C_i^+ = \frac{S_i^-}{S_i^- + S_i^+}, \qquad (i = 1, \ldots, m) \tag{A.8}$$

Partiendo de los resultados de las similitudes obtenidos en nuestro caso, claramente la alternativa preferida es la 2.

C1	0,67
C2	0,83
C3	0,51
C4	0,09
C5	0,45

Tabla A.7: TOPSIS - similitudes o cercanía a la solución ideal positiva

A.3.3. ELECTRE

El método multicriterio ELECTRE trata de establecer cuál es la alternativa preferida estableciendo relaciones de superación entre ellas. Estas relaciones de superación vienen determinadas por la concordancia (conjunto de criterios a favor de A_i);

$$C(A_i, A_k)_j = \begin{cases} 1 & \text{si } r_{ij} \geq r_{kj} \\ & \text{for } (i = 1, \ldots, m;\ j = 1, \ldots, n) \\ 0 & \text{si } r_{ij} < r_{kj} \end{cases} \tag{A.9}$$

y la discordancia (conjunto de criterios a favor de A_k):

$$D(A_k, A_i)_j = \begin{cases} r_{kj} - r_{ij} & \text{si } r_{ij} < r_{kj} \\ & \text{for } (i = 1, \ldots, m;\ j = 1, \ldots, n) \\ 0 & \text{si } r_{ij} \geq r_{kj} \end{cases} \tag{A.10}$$

De estas se definen los coeficientes de concordancia y discordancia.

$$c_{ik} = \sum_{j=1}^{n} C(A_i, A_k)_j \cdot W_j, \qquad (i = 1, \ldots, m;\ j = 1, \ldots, n) \tag{A.11}$$

$$d_{ik} = (\frac{1}{d})\,\text{máx}(D(A_k, A_i)_j), \qquad (i = 1, \ldots, m;\ j = 1, \ldots, n) \tag{A.12}$$

Estos coeficientes se recogen en una matriz (matriz de concordancia y matriz de discordancia) que aplicadas a nuestro problema, quedan:

	C1	C2	C3	C4	C5
A1	1	0,450	0,750	0,900	0,700
A2	0,600	1	1,000	0,900	0,600
A3	0,550	0,650	1	0,900	0,550
A4	0,350	0,100	0,100	1	0,350
A5	0,650	0,400	0,450	0,900	1
A6	0,700	1,000	1,000	1,000	1,000

Tabla A.8: ELECTRE - Matriz de Concordancia

Partiendo de la relación de superación que dice que A_i supera a A_k si $c_{ik} \geq S_c$ y $d_{ik} \leq S_d$, se define como núcleo al subconjunto N (N:subconjunto núcleo, $A-N$: subconjunto no núcleo) de alternativas (A) aquel que:

1. Para cada alternativa $A_k \in A - N$, existe una alternativa $A_i \in N$/A_i S A_k.
2. Para cualesquiera alternativa A_i y $A_k \in N$, ni A_i S A_k, ni A_k S A_i.

	C1	C2	C3	C4	C5
A1	0	1	1	0,16	0,35
A2	0,33	0	0	0,32	0,18
A3	0,43	0,43	0	0,32	0,35
A4	0,67	1	1	0	0,53
A5	0,5	1	1	0,16	0

Tabla A.9: ELECTRE - Matriz de Discordancia

Por lo tanto, el objetivo es seleccionar un subconjunto de alternativas tan restringido como sea posible del conjunto finito de todas las alternativas, evaluado sobre una familia de criterios. Para alcanzarlo se procede a ir ajustando los límites a partir de $Sc = 1$ y discordancia $Sd = 0$, es decir disminuyendo progresivamente el valor de Sc (con límite $Sc = 0$) y aumentando progresivamente el valor de Sd (con límite $Sd = 1$), hasta que se obtenga un la alternativa preferida que será aquella que constituya un núcleo único (subconjuto N de solamente una alternativa). Para realizar esta operación se construye una matriz de superación (tabla A.10) que se obtiene de la matriz de concordancia y discordancia para un Sc y Sd.

Para nuestro caso, el núcleo único se alcanza para un $Sc = 0,65$ y $Sd = 0,29$, valores que consideramos aceptables para tomar una alternativa preferida. Esta se corresponde a la alternativa A2

	A1	A2	A3	A4	A5
A1		0	0	1	0
A2	1		1	1	1
A3	0	0		1	0
A4	0	0	0		0
A5	0	0	0	1	

Tabla A.10: ELECTRE - Matriz de Superación

Apéndice B

Aplicación fórmulas electorales

Este apéndice tiene como misión facilitar la comprensión de las fórmulas electorales distributivas o proporcionales explicadas en el apartado 2.2.2. Para ello se desarrolla una hipotética aplicación, tomada de la bibliografía [CG97], de las variantes de estas fórmulas a un mismo supuesto. Las tablas que se representan en este apartado, también están tomadas de la bibliografía mencionada.

B.1. Características del supuesto

El supuesto utilizado que nos sirve a modo de ejemplo presenta las siguientes características:

- Magnitud de distrito: 8 escaños.
- Votos emitidos: 408.780 votantes.
- Partidos concurrentes: A, B, C, D y E.

En este, el resultado electoral obtenido después de la votación, es el mostrado en la tabla B.1.

Partido	*Nº votos*
A	227.340
B	75.600
C	54.000
D	27.000
E	24.840
TOTAL	408.780

Tabla B.1: Resultado electoral [CG97]

B.2. Resultado electoral

A partir del resultado electoral del supuesto se obtienen las diferentes distribuciones finales de escaños entre partidos, según la variante de la fórmula distributiva aplicada. En los siguientes párrafos se muestran los repartos de escaños para el supuesto planteado.

B.2.1. Fórmulas de cociente electoral común

B.2.1.1. Variante del resto mayor- fórmula Hare

La atribución de escaños para la variante del resto mayor se realiza del siguiente modo:

a) Cálculo de la cuota Hare:

$$Q_h = \frac{V}{M} = \frac{408780}{8} = 51097 \tag{B.1}$$

b) Distribución de escaños según división de votos de cada partido por la cuota y según resto mayor:

Partido	*Nº votos*	*Nº votos/Cuota*	*Escaños según cuota entera*	*Escaños según resto mayor*	*Total escaños por partido*
A	227.340	227.340/51.097 = 4,45	4	0	4
B	75.600	75.600/51.097 = 1,48	1	0	1
C	54.000	54.000/51.097 = 1,06	1	0	1
D	27.000	27.000/51.097 = 0,53	0	1	1
E	24.840	24.840/51.097 = 0,49	0	1	1
TOTAL	408.708		6	2	8

Tabla B.2: Reparto de escaños para la fórmula de cociente entero común con fórmula Hare y media más alta [CG97]

B.2.2. Fórmulas de divisores comunes

La atribución de escaños en este tipo de fórmulas se realiza por medio de los dos siguientes pasos:

a) División del número de votos de cada partido por los divisores: cálculo del cociente.

b) Atribución de los escaños a los partidos que presentan los cocientes mayores hasta proveer los 8 escaños.

Los escaños atribuidos se representan en cursiva en las tablas B.3, B.4, B.5 y B.6 respectivamente para cada variante.

B.2.2.1. Variante d'Hondt

El cociente se calcula a partir del número de votos de cada partido por los divisores 1-2-3-4, etc.

Partido	*Nº votos*	*Nº votos/1*	*Nº votos/2*	*Nº votos/3*	*Nº votos/4*	*Nº votos/5*	*Nº votos/6*	*Nº votos/7*	*Escaños por partido*
A	227.340	*227.340*	*113.670*	*75.780*	*56.835*	*45.468*	*37.890*	32.477	6
B	75.600	*75.600*	37.800	25.200					1
C	54.000	*54.000*	27.000						1
D	27.000	27.000							0
E	24.840	24.840							0

Tabla B.3: Reparto de escaños para la variante d'Hondt [CG97]

B.2.2.2. Variante Sainte-Laguë

El cociente se calcula a partir del número de votos de cada partido por los divisores 1-3-5-7, etc.

Partido	*Nº votos*	*Nº votos/1*	*Nº votos/3*	*Nº votos/5*	*Nº votos/7*	*Nº votos/9*	*Escaños por partido*
						Escaños	
A	227.340	*227.340*	*75.780*	*45.468*	*32.477*	*25.260*	5
B	75.600	*75.600*	25.200	10.800			1
C	54.000	*54.000*	18.000				1
D	27.000	*27.000*	9.000				1
E	24.840	24.840					0

Tabla B.4: Reparto de escaños para la variante Sainte-Laguë [CG97]

B.2.2.3. Variante Sainte-Laguë corregida

El cociente se calcula a partir del número de votos de cada partido por los divisores 1,4-3-5-7, etc.

Partido	*Nº votos*	*Nº votos/1,4*	*Nº votos/3*	*Nº votos/5*	*Nº votos/7*	*Nº votos/9*	*Nº votos/11*	*Escaños por partido*
A	227.340	*162.380*	*75.780*	*45.468*	*32.477*	*25.260*	20.667	5
B	75.600	*54.000*	*25.200*	10.800				2
C	54.000	*38.570*	18.000					1
D	27.000	19.286						0
E	24.840	17.743						0

Tabla B.5: Reparto de escaños para la variante Sainte-Laguë corregida [CG97]

B.2.2.4. Variante Imperiali

El cociente se calcula a partir del número de votos de cada partido por los divisores 2-3-4-5-6-7-8, etc.

Partido	*Nº votos*	*Nº votos/2*	*Nº votos/3*	*Nº votos/4*	*Nº votos/5*	*Nº votos/6*	*Nº votos/7*	*Nº votos/8*	*Escaños por partido*
A	227.340	*113.670*	*75.780*	*56.835*	*45.468*	*37.890*	*32.477*	*28.417*	7
B	75.600	*37.800*	25.200						1
C	54.000	27.000							0
D	27.000	13.500							0
E	24.840	12.420							0

Tabla B.6: Reparto de escaños para la variante Imperiali [CG97]

B.3. Comparación

La tabla B.7 muestra una comparativa de los resultados obtenidos en la sección B.1 de las fórmulas de reparto de escaños escogidas para el supuesto. Como se puede observar la aplicación de las diversas variantes de estas fórmulas puede producir una diferente distribución de escaños.

En esta tabla B.7 se puede observar como, para el supuesto escogido, la fórmula de cociente electoral común (resto mayor con cuota Hare y variante de resto mayor) sería la única que atribuiría escaños a todos los partidos, incluido el partido

menor E, que no obtendría nada bajo ninguna otra fórmula. Entre las fórmulas de divisores comunes, las variantes Sainte-Laguë disminuyen la ventaja del partido mayor A y favorecen a partidos intermedios. A su contra, las variantes d'Hondt e Imperiali son las que muestran un favoritismo hacia los partidos más fuertes (sobre todo la Imperiali), dejando sin representación a los partidos más débiles.

Número de escaños por partido	*Según resto mayor fórmula Hare*	*Según d'Hondt*	*Según Sainte-Laguë*	*Según Sainte-Laguë corregida*	*Según Imperiali*
Partido A	4	6	5	5	7
Partido B	1	1	1	2	1
Partido C	1	1	1	1	0
Partido D	1	0	1	0	0
Partido E	1	0	0	0	0

Tabla B.7: Cuadro-resumen de los resultados del supuesto [CG97]

Apéndice C

GA toolbox

Descrito en la sección 4.1, en este apéndice se incluye el reporte técnico original del software GA toolbox [Sas07].

Además se puede descargar en: `www.illigal.uiuc.edu/web/technical-reports`, con código *IlliGAL Report No. 2007016*

Single and Multiobjective Genetic Algorithm Toolbox in C++

Kumara Sastry

IlliGAL Report No. 2007016
June 2007

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

Single and Multiobjective Genetic Algorithm Toolbox in C++

Kumara Sastry
Illinois Genetic Algorithms Laboratory (IlliGAL),
Materials Computation Center, and
Department of Industrial and Enterprise Systems Engineering
University of Illinois at Urbana-Champaign, Urbana IL 61801

June 5, 2007

Abstract

This report provides documentation for the general purpose genetic algorithm toolbox. The toolbox provides different selection, recombination, mutation, niching, and constraint-handling operators. Problems with single and multiple objectives can be solved with the toolbox. Moreover, the toolbox is easily extensible and customizable for incorporating other operators and for solving user-defined search problems.

1 Introduction

This document briefly describes how to download, compile, and run the GA toolbox. Interested user should refer to the extensive comments in the code for an explanation of the more intricate GA-implementation and the code structure details. This report also explains how to modify the objective function that comes with the distribution of the code. The source is written in C++ but a knowledge of the C programming language is sufficient to modify the objective function so that you can try the toolbox for your own problems.

2 How to download the code?

The code is available from `ftp://ftp-illigal.ge.uiuc.edu/pub/src/GA/GAtoolbox.tgz`. After downloading it, uncompress and untar the file by typing

```
tar zxvf GAtoolbox.tgz
```

At this point you should have in your directory the following files:

```
DISCLAIMER      README          Makefile            chromosome.cpp
chromosome.hpp  crossover.cpp   crossover.hpp       ga.cpp
ga.hpp          globalSetup.cpp globalSetup.hpp     individual.cpp
individual.hpp  localsearch.cpp localsearch.hpp     nsgapopulation.cpp
population.cpp  population.hpp  random.cpp          random.hpp
selection.cpp   selection.hpp   userDefinables.cpp
```

3 How to compile the code?

To compile the code, a C++ compiler needs to be installed on your computer. The code has been compiled using GNU C++ and tested under Linux and Microsoft Visual C++ compiler version 6.0 and GNU C++ compiler distributed under MinGW (`www.mingw.org`) for Windows. To compile the code type `make` on the Linux/Unix shell prompt. After that, you should have an executable file called `GAtbx`.

4 How to run the code?

The executable `GAtbx` needs one argument: the name for an input file. The toolbox reads its parameters from the input file and outputs the results onto the screen (`stdout`). Four sample input files—`input_sga_maxSpec`, `input_sga_minSpec`, `input_nsga_maxSpec`, and `input_nsga_minSpec`—are provided as examples with the distribution of this code.

Two objective functions that comes with the distribution of the code; one is a single objective problem with two constraints and the other is a multiobjective problem with two objectives. The single objective test function is the constrained Himmelblau's function (Rekalaitis, Ravindran, & Ragsdell, 1983). Himmelblau's function is a minimization problem with two decision variables. The objective value for this case is:

$$f(x_1, x_2) = ({x_1}^2 + x_2 - 11)^2 + (x_1 + {x_2}^2 - 7)^2, \tag{1}$$

with the following two constraints

$$(x_1 - 5)^2 + x_2^2 \leq 26, \tag{2}$$

$$4x_1 + x_2 \leq 20. \tag{3}$$

The multiobjective test function is the Kurosawe test function (Deb, Pratap, Agrawal, & Meyarivan, 2002):

$$f_1(x_1, x_2, x_3) = \sum_{i=1}^{2} \left[-10 * \exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2} \right) \right], \tag{4}$$

$$f_2(x_1, x_2, x_3) = \sum_{i=1}^{3} \left[|x_i|^{0.8} + 5 \sin x_i^3 \right]. \tag{5}$$

The steps required to write your own fitness function are described in section 7.

To run the GA toolbox, at the command prompt, type

```
GAtbx <input file name>
```

For example to use `input_sga_maxSpec` as the input file type `GAtbx input_sga_maxSpec`.

Relevant statistics are displayed, depending on the input file specifications, on the screen at the end of each generation. For example, for single-objective optimization problem with niching the output is the population. For multiobjective optimization problem the output is the objective function values of candidate solutions in the population. To change the output, you can modify the function

```
std::ostream &operator<< (std::ostream &out, const Population &pop)
```

implemented in the file `population.cpp`.

5 Description of GA toolbox Capabilities

This section presents an overview of the operators and components implemented in the GA toolbox. The toolbox can solve both single- and multi-objective problems with or without constraints. The multiobjective genetic algorithm implemented is the non-dominated sorting GA II (NSGA-II) (Deb, Pratap, Agrawal, & Meyarivan, 2002).

The decision variables of the problem are encoded as real numbers or as integers within their specified ranges. This encoding procedure permits the decision variables to be binary, discrete, χ-ary alphabet or a real number. The decision variable type needs to be specified in the input file and the valid options are `double` or `int`.

The following genetic operators are supported by GAtbx:

Selection The following selection procedures are available in GA toolbox:

1. Tournament selection with replacement (Goldberg, Korb, & Deb, 1989; Sastry & Goldberg, 2001)
2. Tournament selection without replacement (Goldberg, Korb, & Deb, 1989; Sastry & Goldberg, 2001)
3. Truncation selection (Mühlenbein & Schlierkamp-Voosen, 1993)
4. Roulette-Wheel selection (Goldberg, 1989)
5. Stochastic universal selection (SUS) (Baker, 1985; Baker, 1987; Grefenstette & Baker, 1989; Goldberg, 1989)

Recombination/Crossover The recombination procedures implemented in the GA toolbox are:

1. One point crossover (Goldberg, 1989; Sastry, Goldberg, & Kendall, 2005)
2. Two point crossover (Goldberg, 1989; Sastry, Goldberg, & Kendall, 2005)
3. Uniform crossover (Goldberg, 1989; Sastry, Goldberg, & Kendall, 2005)
4. Simulated binary crossover (SBX) (Deb & Agarwal, 1995; Deb & Kumar, 1995)

Mutation Different mutation procedures implemented in the GA toolbox are:

1. Selective mutation, where a randomly selected gene is replaced by a *uniformly distributed* random value in the interval $[x_{i,min}; x_{i,max}]$ with probability p_m.
2. Genewise mutation, where every gene is mutated using Gaussian mutation with probability p_m.
3. Polynomial mutation (Deb & Agarwal, 1995; Deb & Kumar, 1995; Deb, 2001).

Niching Niching methods implemented in the GA toolbox are:

1. Fitness sharing (Goldberg & Richardson, 1987; Goldberg, 1989)
2. Deterministic crowding (Mahfoud, 1992)
3. Restricted tournament selection (Harik, 1995)

Scaling Three scaling procedures are implemented in the GA toolbox:

1. Linear ranking (Baker, 1985; Goldberg, 1989),
2. Sigma scaling (Forrest, 1985; Goldberg, 1989), and

3. Default scaling which is only used if any one of the following conditions hold true: (a) The selection procedure is roulette-wheel or SUS, and the minimum fitness value is negative, or (b) the niching method is fitness sharing and the minimum fitness value is negative.

Constraint handling Three constraint handling methods are implemented in the GA toolbox:

1. Tournament method (Deb, 2001)
2. Linear penalty method, where a weighted sum of the constraint violations of an individual is added to or subtracted from its objective value depending on whether we are minimizing or maximizing the objective function.
3. Quadratic penalty method, which is similar to linear penalty, except that the penalty value is computed as a weighted sum of the square of constraint violation values of a candidate solution.

It is important to note that the constraint handling methods use the constraint violation value and not constraint function value. For example, if the constraint is

$$10x_1 + x_2 \leq 20,$$

$x_1 = 2$, and $x_2 = 2$, the constraint violation value is 2. However, if $x_1 = 1$, $x_2 = 7$, then the constraint violation value is 0.

Local search The local search method implemented in the GA toolbox is the *Nelder-Mead* algorithm (Nelder & Mead, 1965; Press, Flannery, Teukolsky, & Vettering, 1989). Local search is applied to an individual with probability p_{ls}. The solution obtained through the local search is incorporated using the *Baldwinian strategy*—where the local search solution replaces the individual, but the fitness of the individual is not modified—with probability, p_b, and *Lamarckian strategy*—where both the fitness and the individual are replaced with local search solution and its fitness—with probability, $1 - p_b$.

Elitist Replacement In GA toolbox, both the old and new population is sorted according to their fitness and constraint violation. A user specified proportion, p_e, of top individuals are retained and the rest, $n(1 - p_e)$ individuals are replaced by the top individuals in the new population. Here n is the population size.

5.1 Statistics

Various statistics are collected from the population every generation. These not only aid the user with the progress of genetic search, but are also used in many operators and functionalities. Statistics are also used to determine if the genetic search should be terminated or not. The statistics collected are the mean, minimum, and maximum fitness and objective values. Other statistics include, fitness variance and objective variance in the population. For the multiobjective case, the statistics are collected separately for each objective.

5.2 Stopping Criteria

GA toolbox provides the following stopping criteria for the GA runs

1. **Number of Function Evaluations**: This option terminates the run after certain number of function evaluations have been performed by the GA.

2. **Fitness Variance**: This option stops the once the fitness variance of the population reaches below a specified value.
3. **Best Fitness**: This criterion terminates the run once the fitness of the best individual in the population exceeds a specified value.
4. **Average Fitness**: This option stops the run once the average fitness of the population exceeds a specified value.
5. **Average Objective**: This option stops the run once the average objective function value of the population exceeds a specified value.
6. **Change in Best Fitness**: This option terminates the run once the change in fitness of the best individual in the population over the previous generation is below a specified value.
7. **Change in Average Fitness**: This option terminates the run once the change in average fitness of the population over the previous generation is below a specified value.
8. **Change in Fitness Variance**: This option terminates the run once the change in fitness variance of the population over the previous generation is below a specified value.
9. **Change in Best Objective**: This option terminates the run once the change in change in objective value of the best individual in the population over the previous generation is below a specified value.
10. **Change in Average Objective**: This option terminates the run once the change in change in average objective value the population over the previous generation is below a specified value.
11. **Number of Fronts**: This option is available only for NSGA. It terminates the run after the number of non-dominated fronts in the population goes below a specified value.
12. **Number of guys in first front**: This option terminates the run when the number of individuals in the first exceeds a specified value. It is only available only for NSGA.
13. **Change in number of fronts**: This options stops the run when the change in the number of fronts over the previous generating is below a specified value.

6 Input File Format

The input file `input_sga_maxSpec` is shown (along with line numbers) below. As mentioned earlier, the `GAtbx` skips all empty lines and those that start with `#`. Then it starts reading the parameters in a predefined order. The program doesn't do any fancy parsing on the input file. This means that **you should not change the order of the lines in the input file** and therefore the choice of operators and parameters will be incorrectly read. The input file is self explanatory and straightforward to understand.

```
1    #
2    # GA type: SGA or NSGA
3    #
4    SGA
```

```
5
6    #
7    # Number of decision variables
8    #
9    2
10
11   #
12   # For each decision variable, enter:
13   #   decision variable type, Lower bound, Upper bound
14   # Decision variable type can be double or int
15   #
16   double 0.0 6.0
17   double 0.0 6.0
18
19   #
20   # Objectives:
21   #   Number of objectives
22   #   For each objective enter the optimization type: Max or Min
23   #
24   1
25   Min
26
27   #
28   # Constraints:
29   #   Number of constraints
30   #   For each constraint enter a penalty weight
31   #
32   2
33   1.0
34   1.0
35   #
36   # General parameters: If these parameters are not entered default
37   #                     values will be chosen. However you must enter
38   #                     "default" in the place of the parameter.
40   #   [population size]
41   #   [maximum generations]
42   #   [replace proportion]
43   #
44   100
45   100
46   0.9
47
48   #
49   # Niching (for maintaining multiple solutions)
50   # To use default setting type "default"
51   #  Usage: Niching type, [parameter(s)...]
52   #  Valid Niching types and optional parameters are:
53   #   NoNiching
```

```
54  #   Sharing [niching radius] [scaling factor]
55  #   RTS [Window size]
56  #   DeterministicCrowding
57  #
58  #  When using NSGA, it must be NoNiching (OFF).
59  #
60  NoNiching
61
62  #
63  # Selection
64  #  Usage: Selection type, [parameter(s)...]
65  #  To use the default setting type "default"
66  #
67  #  Valid selection types and optional parameters are:
68  #   RouletteWheel
69  #   SUS
70  #   TournamentWOR [tournament size]
71  #   TournamentWR [tournament size]
72  #   Truncation [# copies]
73  #
74  #  When using NSGA, it can be neither SUS nor RouletteWheel.
75  #
76  TournamentWOR 2
77
78  #
79  # Crossover
80  #  Crossover probability
81  #  To use the default setting type "default"
82  #
83  #  Usage: Crossover type, [parameter(s)...]
84  #  To use the default crossover method type "default"
85  #  Valid crossover types and optional parameters are
86  #       OnePoint
87  #       TwoPoint
88  #       Uniform [genewise swap probability]
89  #       SBX [genewise swap probability][order of the polynomial]
90  #
91  0.9
92  SBX 0.5 10
93
94  #
95  # Mutation
96  #  Mutation probability
97  #  To use the default setting type "default"
98  #
99  #  Usage: Mutation type, [parameter(s)...]
100 #  Valid mutation types and the optional parameters are:
101 #       Selective
```

```
102 #       Polynomial [order of the polynomial]
103 #       Genewise [sigma for gene #1][sigma for gene #2]...[sigma for gene #ell]
104 #
105 0.1
106 Polynomial 20
107
108 #
109 # Scaling method
110 #  To use the default setting type "default"
111 #
112 #  Usage: Scaling method, [parameter(s)...]
113 #  Valid scaling methods and optional parameters are:
114 #       NoScaling
115 #       Ranking
116 #       SigmaScaling [scaling parameter]
117 #
118 NoScaling
119
120 #
121 # Constraint-handling method
122 # To use the default setting type "default"
123 #
124 # Usage: Constraint handling method, [parameters(s)...]
125 # Valid constraint handling methods and optional parameters are
126 #       NoConstraints
127 #       Tournament
128 #       Penalty [Linear|Quadratic]
129 #
130 Tournament
131
132 #
133 # Local search method
134 # To use the default setting type "default"
135 #
136 # Usage: localSearchMethod, [maxLocalTolerance], [maxLocalEvaluations],
137 #               [initialLocalPenaltyParameter], [localUpdateParameter],
138 #               [lamarckianProbability], [localSearchProbability]
139 #
140 # Valid local search methods are: NoLocalSearch and SimplexSearch
141 #
142 # For example, SimplexSearch 0.001000 20 0.500000 2.000000 0.000000 0.000000
143 NoLocalSearch
144
145 #
146 # Stopping criteria
147 # To use the default setting type "default"
148 #
149 # Number of stopping criterias
```

```
150  #
151  # If the number is greater than zero
152  #    Number of generation window
153  #    Stopping criterion, Criterion parameter
154  #
155  # Valid stopping criterias and the associated parameters are
156  #       NoOfEvaluations, Maximum number of function evaluations
157  #       FitnessVariance, Minimum fitness variance
158  #       AverageFitness, Maximum value
159  #       AverageObjective, Max/Min value
160  #       ChangeInBestFitness, Minimum change
161  #       ChangeInAvgFitness, Minimum change
162  #       ChangeInFitnessVar, Minimum change
163  #       ChangeInBestObjective, Minimum change
164  #       ChangeInAvgObjective, Minimum change
165  #       NoOfFronts (NSGA only), Minimum number
166  #       NoOfGuysInFirstFront (NSGA only), Minimum number
167  #       ChangeInNoOfFronts (NSGA only), Minimum change
168  #       BestFitness (SGA with NoNiching only), Maximum value
169  #
170  0
171
172  #
173  # Load the initial population from a file or not
174  # To use the default setting type "default"
175  #
176  # Usage: Load population (0|1)
177  #
178  # For example, if you want random initialization type 0
179  # On the other and if you want to load the initial population from a
180  # file, type
181  #       1 <population file name> [0|1]
182  #
183  # Valid options for "Load population" are 0/1
184  # If you type "1" you must specify the name of the file to load the
185  # population from. The second optional parameter which indicates
186  # whether to evaluate the individuals of the loaded population or not.
187  0
188
189  # Save the evaluated individuals to a file
190  #
191  # To use default setting type "default".
192  #
193  # Here by default all evaluated individuals are stored and you will be
194  # asked for a file name later when you run the executable.
195  #
196  # Usage: Save population (0|1)
197  # For example, if you don't want to save the evaluated solutions type 0
```

```
198  # On the other and if you want to save the evaluated solutions
199  #       1 <save file name>
200  #
201  # Note that the evaluated solutions will be appended to the file.
202  #
203  # Valid options for "Save population" are 0/1
204  # If you type "1" you must specify the name of the file to save the
205  # population to.
206  1 evaluatedSolutions.txt
207
208  #END
```

---

7 How to plug-in your own objective function?

The code for the objective function is in the file `userDefinables.cpp`. This is the only file that you need to rewrite in order to try your own fitness function. The function header is as follows:

```
void globalEvaluate(double *x, double *objArray, double *constraintViolation,
                    double *penalty, int *noOfViolations)
```

It takes as argument an array `x`, whose ℓ elements contains the decision variables of a candidate solution whose fitness is being evaluated. Here, ℓ is the problem length (# of genes). The objective function value(s) is(are) returned in the array `objArray`, the constraint violation value(s) in the array `constraintViolation`, penalty to be added to the fitness function in `penalty` (used only if the linear or quadratic penalty methods are chosen) constraint handling methods are chosen)and number of constraints violated in `noOfViolations`.

8 About the C++ code

The implementation of the GA toolbox doesn't use advanced features of the C++ language such as templates and inheritance. This means that you don't need to be a C++ expert in order to modify the code. In fact, you can modify the code and plug-in your own objective function using the C programming language alone. Next, we give brief description of the source files. Each `.cpp` file has a corresponding `.hpp` file, except `nsgapopulation.cpp` and `userDefinables.cpp`. The `.hpp` files are the header files and contain the definitions of the various classes. The `.cpp` files contain the actual implementation.

`chromosome.cpp` contains the implementation of the class `chromosome`. A chromosome is an array of genes. It also contains implementation of mutation operators.

`individual.cpp` contains the implementation of the class `individual`. An individual is a candidate solution for a given search and optimization problem.

`population.cpp` contains the implementation of the class `population`. A population is an array of chromosomes. Objective to fitness mappings and statistics computations are implemented here.

`nsgapopulation.cpp` contains the implementation of the class `nsgapopulation`. It builds on the class `population` and implements nondominated sorting and crowding distance computation required for NSGA-II.

`crossover.cpp` contains the implementation of the class `crossover`. Different crossover operators are implemented here.

`selection.cpp` contains the implementation of the class `selection`. Different selection operators are implemented here.

`localsearch.cpp` contains the implementation of the class `localsearch`. A local search operator is implemented here.

`ga.cpp` contains the implementation of the class `ga`. Different GA architectures (GA and NSGA-II) are implemented here.

`random.cpp` contains subroutines related to the pseudo random number generator.

`globalSetup.cpp` contains the implementation of the class `globalSetup`. Global data needed across different classes are implemented here.

`userDefinables.cpp` contains the code for the objective function. If you want to try the GAtbx on your own problem, you should modify the function `globalEvaluate()` contained in this file.

9 Disclaimer

This code is distributed for academic purposes only. It has no warranty implied or given, and the authors assume no liability for damage resulting from its use or misuse. If you have any comments or find any bugs, please send an email to `kumara@illigal.ge.uiuc.edu`.

10 Commercial use

For the commercial use of this code please contact Prof. David E. Goldberg at `deg@uiuc.edu`

Acknowledgments

This work was also sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, the National Science Foundation under grant ITR grant DMR-03-25939 at the Materials Computation Center. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

I thank Prof. Duane D. Johnson and David E. Goldberg for encouraging me to write this report.

References

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, 101–111.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: John Wiley and Sons.

Deb, K., & Agarwal, R. (1995). Simulated binary crossover for continuous search space. *Complex Systems*, *9*, 115–148.

Deb, K., & Kumar, A. (1995). Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems. *Complex Systems*, *9*, 431–454.

Deb, K., Pratap, A., Agrawal, S., & Meyarivan, T. (2002). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. (Also KanGAL Report No. 2000001).

Forrest, S. (1985). *Documentation fo PRISONERS DILEMMA and NORMS programs that use genetic algorithms*. Unpublished manuscript, University of Michichan, Ann Arbor.

Goldberg, D. E. (1989). *Genetic algorithms in search optimization and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530. (Also IlliGAL Report No. 89003).

Goldberg, D. E., & Richardson, J. J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proceedings of the Second International Conference on Genetic Algorithms*, 41–49.

Grefenstette, J. J., & Baker, J. E. (1989). How genetic algorithms work: A critical look at implicit parallelism. *Proceedings of the Third International Conference on Genetic Algorithms*, 20–27.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *Proceedings of the Sixth International Conference on Genetic Algorithms*, 24–31. (Also IlliGAL Report No. 94002).

Mahfoud, S. W. (1992). Crowding and preselection revisited. *Parallel Problem Solving from Nature*, *2*, 27–36. (Also IlliGAL Report No. 92004).

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. continous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, *8*, 308–313.

Press, W., Flannery, B., Teukolsky, S., & Vettering, W. (1989). *Numerical recipes in C*. Cambridge: Cambridge University Press.

Rekalaitis, G., Ravindran, A., & Ragsdell, K. (1983). *Engineering optimization: Methods and applications*. New York, NY: Wiley.

Sastry, K., & Goldberg, D. E. (2001). Modeling tournament selection with replacement using apparent added noise. *Intelligent Engineering Systems Through Artificial Neural Networks*, *11*, 129–134. (Also IlliGAL Report No. 2001014).

Sastry, K., Goldberg, D. E., & Kendall, G. (2005). Genetic algorithms: A tutorial. In Burke, E., & Kendall, G. (Eds.), *Introductory Tutorials in Optimization, Search, and Decision Support Methodologies* (Chapter 4, pp. 97–125). Berlin: Springer. http://www.asap.cs.nott.ac.uk/publications/pdf/gxk_introsch4.pdf.

Apéndice D

Funciones test y procedimientos de evaluación

Para la selección de la batería de funciones test y el procedimiento de evaluación que figura en el apartado 3.2.1 se contaron con diferentes alternativas buscadas en la red y la bibliografía. En este apéndice se recogen estas alternativas además de describirse las características y aspectos más importantes, así como los problemas que pueden presentar.

D.1. Introducción

Los estudios basados en el análisis según un procedimiento de evaluación sobre un conjunto de funciones test son necesarios para cuantificar la efectividad de diferentes algoritmos genéticos en problemas de optimización. Existen algunos casos en el que se presenta conjuntamente una batería de *funciones de prueba* o *funciones test* (test functions) y de un *procedimiento de evaluación* (evaluation criterias), al que se denomina test suite. En un estudio con una batería de funciones test y un procedimiento de evaluación determinado se debe:

- Poder evaluar adecuadamente el algoritmo en pruebas bajo diferentes tipos de escenarios.
- Permitir la comparativa de diferentes algoritmos de optimización.
- Mostrar la evolución del algoritmo en tiempo / nº evaluaciones con respecto al tamaño del problema.

En las siguientes secciones se describen las características y aspectos más importantes que debe recoger las test suites, para obtener así unos buenos resultados.

D.1.0.5. Batería de funciones test

En la mayoría de los estudios sobre computación evolutiva se considera un pequeño conjunto de problemas estándar como prueba. Sin embargo, algunas comparaciones presentadas en algunos estudios son a menudo confusas y limitadas en relación a las funciones de prueba utilizadas. En ocasiones, las funciones test y el algoritmo escogido, son complementarios y este mismo algoritmo no proporciona resultados satisfactorios con otros problemas. Por lo tanto, se debe disponer de una completa batería de funciones test (conjunto de funciones test), que no presente estos problemas, homogénea, con un número aceptable de funciones y que recoja las características que se describen a continuación.

Características que debe reunir el conjunto de funciones:

1. En todos los casos se busca minimizar la función objetivo.
2. Funciones continuas / discontinuas.
3. Funciones convexas / no convexas.
4. Funciones unimodales / multimodales.
5. Funciones cuadráticas / no cuadráticas
6. Funciones con baja y alta dimensionalidad, es decir que puedan manejar baja y alta cantidad de variables de decisión.
7. Funciones determinísticas / estocásticas.
8. Funciones con ruido.

D.1.0.6. Procedimiento de evaluación

Una vez definida una batería de funciones test, acorde con las especificaciones requeridas, resulta necesario poder analizar los resultados que se derivan de los posibles estudios sobre estas funciones test en diversos escenarios. Por lo tanto la idea básica es *cuantificar los resultados de las funciones test*. Se debería incluir en este procedimiento de evaluación:

- Convergencia hacia un óptimo local/global, como por ejemplo número de evaluaciones para alcanzar un óptimo.
- Rendimiento evaluación tras evaluación (mejora en curso).
- Diferencia en la finalización con el óptimo global (conocido).

D.2. Problemas en funciones test con restricciones

Ciertas características de las funciones test pueden hacer que el resultado no sea el esperado o irreal para algunos algoritmos [SK06]. Por ejemplo, muchas funciones test definen el óptimo global en el origen. Esto implica que todas las variables toman el mismo valor numérico, cero, en el óptimo global. Otro ejemplo es que todas las variables presenten el mismo intervalo de valores. A continuación se enumeran algunas de las características más comunes:

- El óptimo global presenta los mismos valores de los parámetros en las distintas variables/dimensiones.
- Óptimo global en el origen.
- Óptimo global situado en el centro de un rango de búsqueda.
- Óptimo global situado en la frontera.
- Óptimo local situado a lo largo de los ejes de coordenadas.
- Patrón repetitivo en el espacio de búsqueda.

Por lo tanto, se puede dar el caso de que un algoritmo que haya presentado buenos resultados en un estudio con un determinado conjunto de funciones test, al llevarlo a resolver problemas de optimización pueda presentar algunas deficiencias que quedaron ocultas ante alguna de los puntos anteriormente enumerado o simplemente se deteriore. Partiendo de esto, habrá que asegurarse que la batería de funciones contemple estos problemas y los aborde de algún modo.

Algunas técnicas para resolverlo A continuación se describen algunas posibles técnicas desarrolladas para resolverlo:

- Desplazando el óptimo global a alguna posición aleatoria hace que el este óptimo global presente diferentes valores de las variables $[x_1, x_2, x_3, \ldots, x_n]$ para diferentes dimensiones.
- Rotar la función según se muestra:

$$F(\mathbf{x}) = f(R \cdot \mathbf{x}); \tag{D.1}$$

donde: R matriz ortogonal

- Usar diferentes tipos de funciones y diferentes matrices de rotación para componer un solo problema test.
- Mezclar en una función test compuesta diferentes propiedades de diferentes funciones test básicas para destruir estructuras repetitivas.

D.3. Baterías de funciones, procedimientos de evaluación y test suites

A continuación se muestran las opciones encontradas, en la bibliografía o en la red, de baterías de funciones test (test functions), procedimientos de evaluación y de test suites, con las que se ha contado para la selección (ver apartado 3.2.1).

D.3.1. Estudio de De Jong

En 1975, De Jong [Gol89a] completó su estudio *An Analysis of the Behavior of Calss of Genetic Adaptative Systems*. En él consideró a los algoritmos genéticos en el marco de la optimización de funciones aunque era consciente de las posibles aplicaciones de los algoritmos genéticos en otros campos, tales como el diseño de estructuras de datos, diseño de algoritmos y el control adaptativo del sistema operativo de computadores.

De Jong diseñó una batería de problemas que constaba de 5 funciones de minimización, en las que prestó especial atención para que el conjunto englobara las siguientes características:

- Continuas/discontinuas
- Convexas/no convexas
- Unimodales/modales
- Cuadráticas/no cuadráticas
- Baja dimensión/alta dimensión
- Deterministas/estocásticas

La tabla D.1 recoge las 5 funciones y sus intervalos en los que se aplican. De la figura D.1 a la D.5 se muestra la representación *invertida* de estas.

Función	*Expresión*	*Límites*
1	$f(x_i) = \sum_{i=1}^{3} {x_i}^2$	$x \in [-5{,}12;\ 5{,}12]$
2	$f(x_1, x_2) = 100 \cdot ({x_1}^2 - x_2)^2 + (1 - x_1)^2$	$x \in [-2{,}048;\ 2{,}048]$
3	$f(x_i) = \sum_{i=1}^{5} integer(x_i)$	$x \in [5{,}12;\ 5{,}12]$
4	$f(x_i) = \sum_{i=0}^{30} x_j^4 + gauss(0,1)$	$x \in [-1{,}28;\ 1{,}28]$
5	$f(x_i) = 0{,}002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$	$x \in [-65{,}536;\ 65{,}536]$

Tabla D.1: Cinco funciones de De Jong [Gol89a]

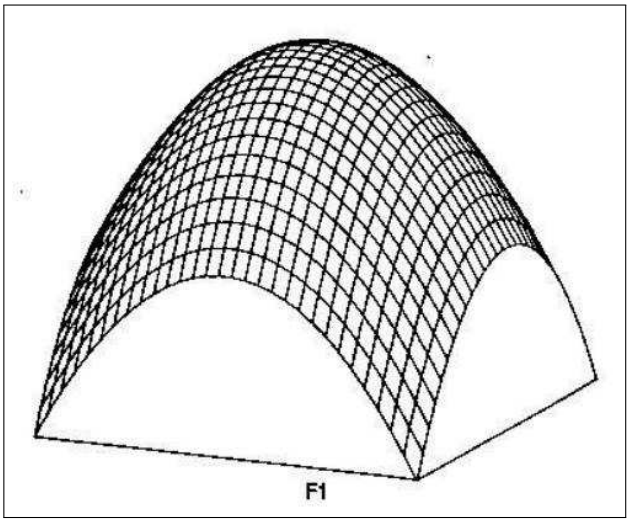

Figura D.1: Representación invertida 3D de la función F1 de De Jong [Gol89a]

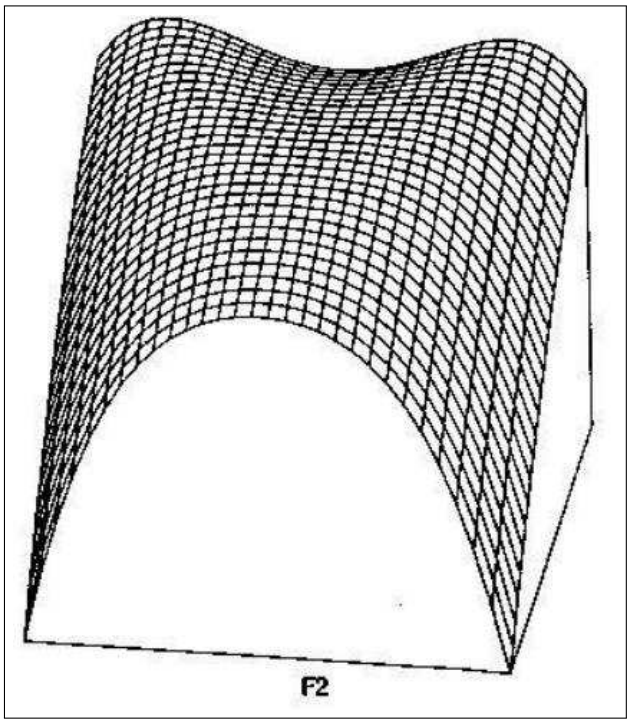

Figura D.2: Representación invertida 3D de la función F2 de De Jong [Gol89a]

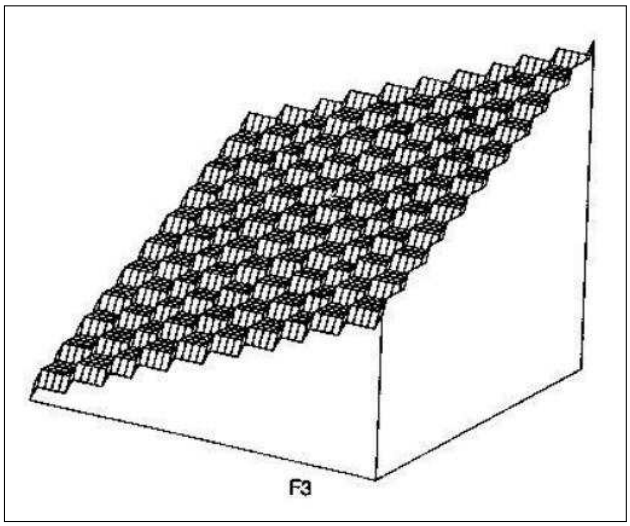

Figura D.3: Representación invertida 3D de la función F3 de De Jong [Gol89a]

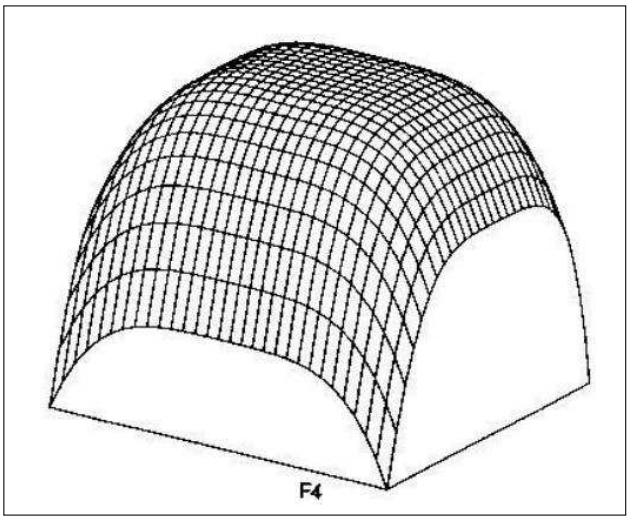

Figura D.4: Representación invertida 3D de la función F4 de De Jong [Gol89a]

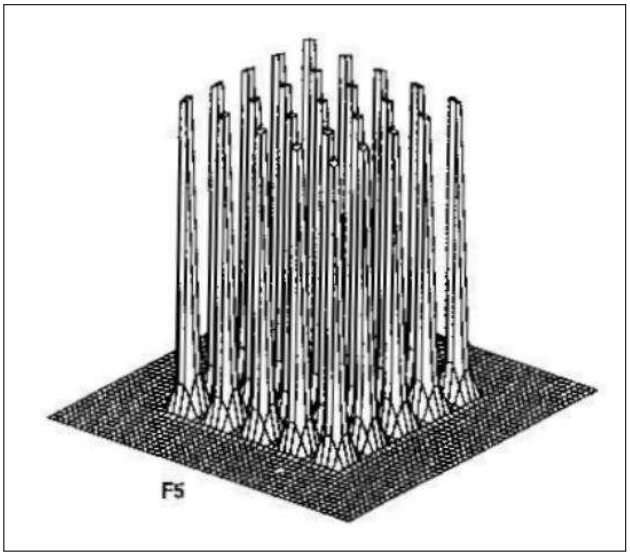

Figura D.5: Representación invertida 3D de la función F5 de De Jong [Gol89a]

Para cuantificar la efectividad de los diferentes algoritmos genéticos, De Jong inventó dos medias a las que llamó off-line y on-line.

En una aplicación off-line, se pueden simular una gran cantidad de evaluaciones de la función y la mejor alternativa se salva para ser usada en una aplicación posterior. En una on-line, se almacenan las evaluaciones por experimentación.

En su estudio, De Jong definió el coeficiente on-line $x_e(s)$ de la estrategia s bajo las condiciones e como:

$$x_e(s) = \frac{1}{T}\sum_{1}^{T} f_e(t), \tag{D.2}$$

donde $f_e(t)$ es el valor de la función objetivo bajo las condiciones e en la iteración t.

El coeficiente off-line lo definió como:

$$x_e^*(s) = \frac{1}{T}\sum_{1}^{T} f_e^*(t), \tag{D.3}$$

donde $f_e^* =$ mejor de $(f_e(1), f_e(2), \ldots, f_e(t))$.

En otras palabras, el coeficiente off-line mide el promedio de los mejores valores procesados hasta una determinada iteración.

D.3.2. GATbx: Genetic Algorithm Toolbox for use with MATLAB

En el documento de la bibliografía "Genetic Algorithm Toolbox Test Functions" de Hartmut Pohlheim [Poh06] se describen algunas de las funciones test implementadas en el GATbx Toolbox de algoritmos genéticos para Matlab. Estas funciones están recogidas de diferentes literaturas de algoritmos genéticos, estrategias de evolución y optimización global. La tabla D.2 muestra el conjunto escogido para este documento, de las cuales las 8 primeras se corresponden con típicos problemas paramétricos y el resto con problemas de optimización dinámicos.

No.	*Función*
1	De Jong's function 1
2	Axis parallel Hyper-ellipsoid
3	Rotated Hyper-ellipsoid
4	Rosenbrock's valley (banana function)
5	Rastrigins's function
6	Schwefel's function
7	Griewangk's function
8	Sum of different Powers
9	Double Integrator
10	Harvest problem
11	Linear-quadratic problem (Discret Function)
12	Linear-quadratic problem (Continious Function)
13	Push-cart problem

Tabla D.2: Set de funciones test detalladas en [Poh94]

Más ampliamente, Hartmut Pohlheim, el autor de este software, describe y representa en la página web [1] (donde también se pueden obtener dicho software, sus características y tutoriales) el conjunto completo de funciones test "de ejemplo" que se disponen. En total son 16, cuya representación (para 2 variables) y características han sido sacadas del [Poh06] y se encuentran detalladas a continuación:

De Jong's function 1 Se trata de la función más simple del test de De Jong, conocida también como función Esfera. Unimodal, continua y convexa.

$$f_1(x_i) = \sum_{i=1}^{n} x_i^2; \qquad x_i \in [-5{,}12;\ 5{,}12] \tag{D.4}$$

[1] http://www.geatbx.com/docu/fcnindex-01.html citepohlheim$_url$

mínimo global: $x_i = 0$ $f(x) = 0$

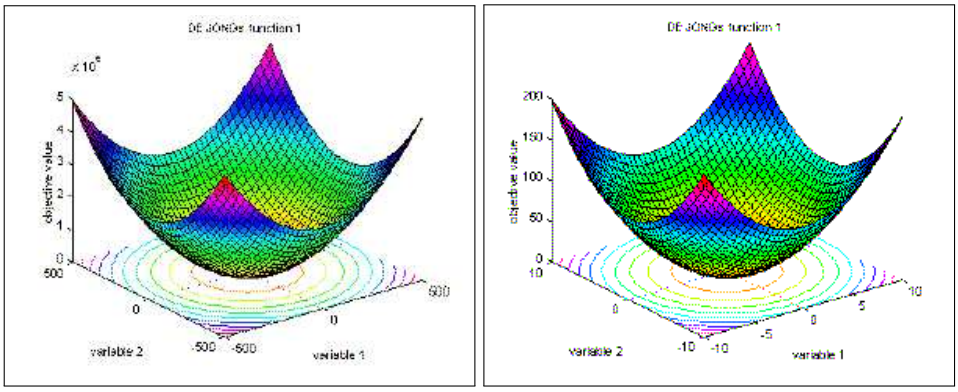

Figura D.6: Representación de la función 1 de De Jong para diferentes dominios ($[-500, 500] y [-10, 10]$) [Poh06]. A pesar del cambio de escala, ambas gráficas conservan el mismo aspecto

Axis parallel Hyper-ellipsoid Función similar a la anterior (De Jong 1). También conocida como función Esfera con peso, se trata de una función unimodal, continua y convexa.

$$f_2(x_i) = \sum_{i=1}^{n} i \cdot {x_i}^2; \qquad x_i \in [-5{,}12;\ 5{,}12] \tag{D.5}$$

mínimo global: $x_i = 0$ $f(x) = 0$

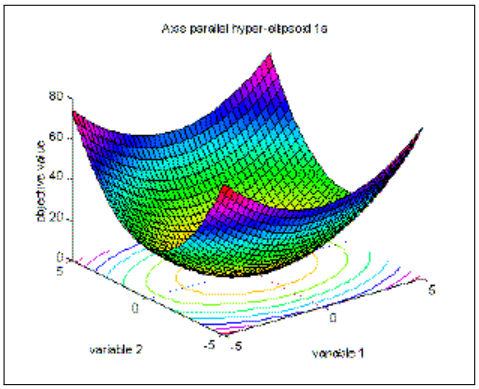

Figura D.7: Representación de la función axis parallel Hyper-ellipsoid para $[-5, 5]$ [Poh06]

Rotated Hyper-ellipsoid Una extensión de la *axis parallel Hyper-ellipsoid* es la función de Schwefel 1.2. Con respecto a los ejes de coordenadas, esta función produce hyper-elipsoides rotadas. Función continua, convexa y unimodal.

$$f_3(x_i) = \sum_{i=1}^{n} \left(\sum_{i=1}^{i} x_i \right)^2; \qquad x \in [-65{,}536;\ 65{,}536] \tag{D.6}$$

mínimo global: $x_i = 0$ $f(x) = 0$

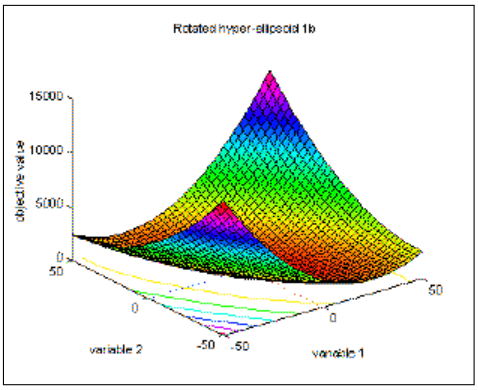

Figura D.8: Representación de la función rotated Hyper-ellipsoid para $[-50, 50]$ [Poh06]

Moved axis parallel Hyper-ellipsoid function Función derivada de la *axis parallel Hyper-ellipsoid*. A diferencia de la anterior, esta función presenta una forma más elíptica. Además el mínimo de la función ya no se encuentra en $x_i = 0$.

$$f_4(x_i) = \sum_{i=1}^{n} 5 \cdot i \cdot {x_i}^2; \qquad x_i \in [-5{,}12;\ 5{,}12] \tag{D.7}$$

mínimo global: $x_i = 5 \cdot i$ $f(x) = 0$

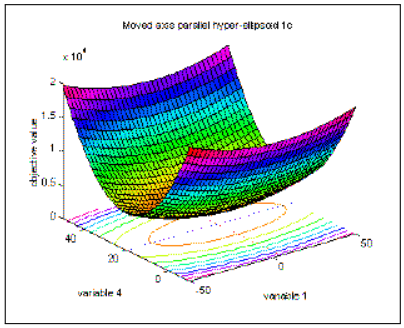

Figura D.9: Representación de la función moved axis parallel Hyper-ellipsoid para $[-50, 50]$ [Poh06]

Rosenbrock's valley (De Jong's function 2) Típico problema de optimización muy usado en el diseño de algoritmos. El óptimo global se ubica en un plano valle pero con forma parabólica, alargada y estrecha; por lo que encontrar el valle es relativamente sencillo y sin embargo la dificultad radica en converger al óptimo.

$$f_5(x_i) = \sum_{i=1}^{n-1} 100 \cdot \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2; \qquad x_i \in [-2{,}048;\ 2{,}048] \quad \text{(D.8)}$$

$$\text{mínimo global:} \qquad x_i = 1 \qquad f(x) = 0$$

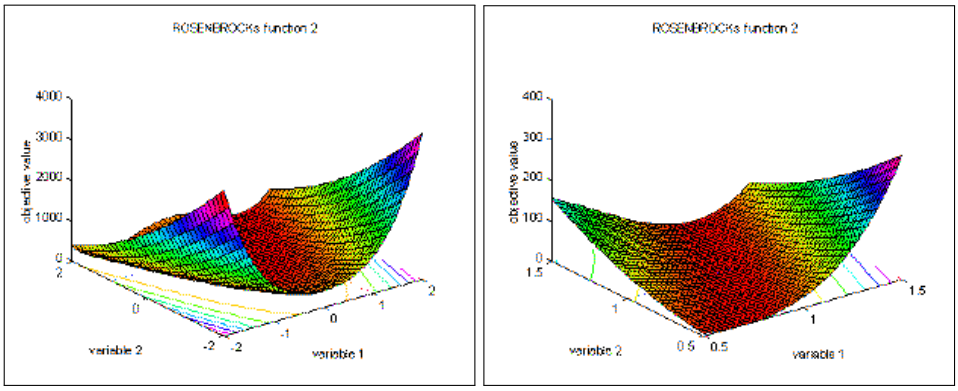

Figura D.10: Representación de la función de Rosenbrock [Poh06]. A la izquierda el rango completo ($[-2, 2]$), a la derecha el área alrededor del óptimo situado en $[1, 1]$

Rastrigin's function Esta función está basada en la primera, con la adicción de una componente senoidal, para conseguir así un gran número de óptimos locales. Debido a esto, la función es altamente multimodal con los mínimos locales regularmente distribuidos.

$$f_6(x_i) = 10 \cdot n \sum_{i=1}^{n} x_i^2 - 10 \cdot cos(2 \cdot \pi \cdot x_i); \qquad x_i \in [-5{,}12;\ 5{,}12] \quad \text{(D.9)}$$

$$\text{mínimo global:} \qquad x_i = 0 \qquad f(x) = 0$$

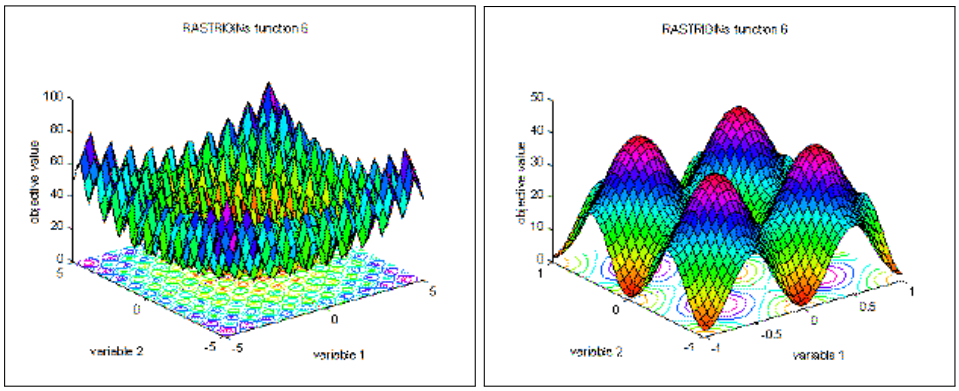

Figura D.11: Representación de la función de Rastrigin [Poh06]. A la izquierda representada en un área de ($[-5, 5]$), a la derecha alrededor del óptimo situado en $[0, 0]$

Schwefel's function La función de Schwefel es engañosa en relación a que el mínimo global está distanciado en el espacio de búsqueda del mejor mínimo local. Por lo tanto, los algoritmos genéticos son propensos a converger en la dirección equivocada.

$$f_7(x_i) = \sum_{i=1}^{n} -x_i \cdot \text{sen}\,(\sqrt{x_i}); \qquad x_i \in [-500;\ 500] \tag{D.10}$$

$$\text{mínimo global:} \qquad x_i = 420{,}9687 \qquad f(x) = n \cdot 418{,}9829$$

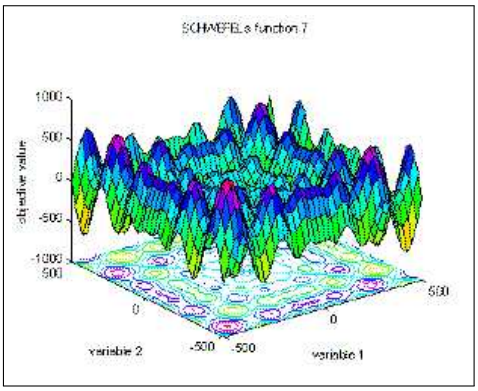

Figura D.12: Representación de la función de Shwefel para $[-500, 500]$ [Poh06]

Griewangk's function La función de Griewangk es similar a la de Rastrigin ya que muestra un gran número de mínimos locales a lo largo y ancho del espacio de

búsqueda, regularmente distribuidos.

$$f_8(x_i) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \frac{\cos(x_i)}{\sqrt{i}}; \qquad x_i \in [-600;\ 600] \qquad \text{(D.11)}$$

$$\text{mínimo global:} \qquad x_i = 0 \qquad f(x) = 0$$

La figura D.13 representa la función de Griewangk para diferentes resoluciones, representando cada una de estas gráficas diferentes propiedades de la función. La gráfica superior izquierda muestra el rango de definición completo. Aquí, la función tiene un aspecto muy similar al de la función 1 de De Jong. Cuando nos aproximamos al área interna, la función cambia considerablemente de aspecto. Aparece un gran número de picos y valles, como se puede ver en la gráfica superior derecha. Aplicando un zoom al área del óptimo (gráfica inferior) los picos y valles se suavizan.

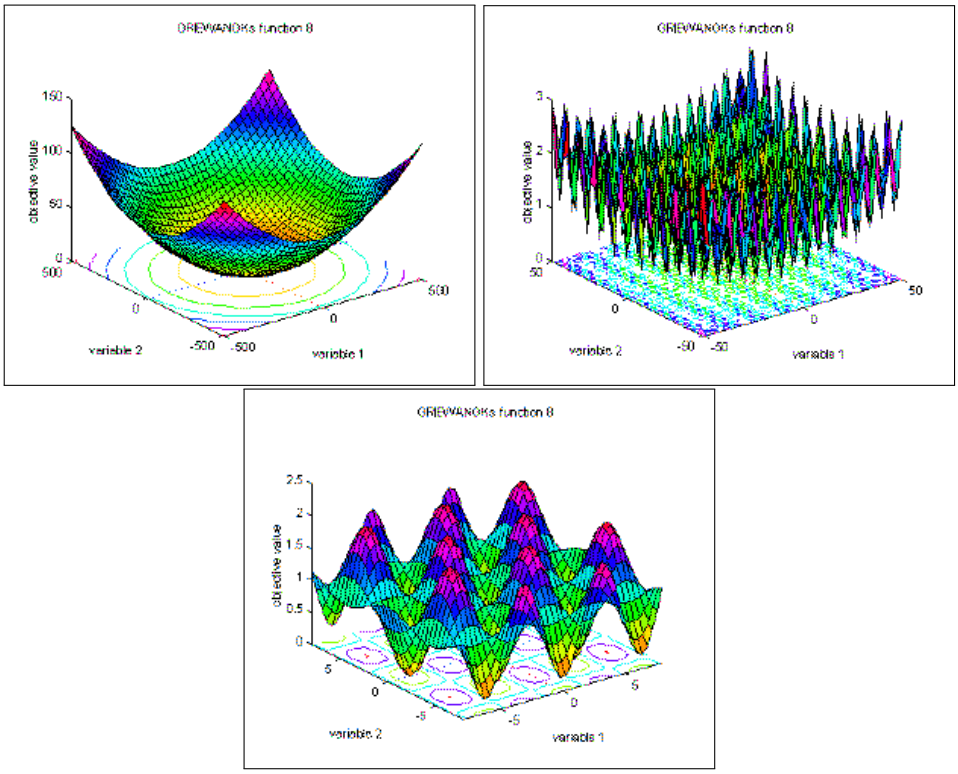

Figura D.13: Representación de la función de Griewangk [Poh06]. Arriba a la izquierda el área completa $[-500, 500]$, arriba a la derecha para $[-50, 50]$ (zoom) y abajo alrededor del óptimo situado en $[0, 0]$

Sum of different Powers Se trata de una función unimodal frecuentemente usada.

$$f_9(x_i) = \sum_{i=1}^{n} \|x_i\|^{i+1}; \qquad x_i \in [-1;\ 1] \tag{D.12}$$

mínimo global: $x_i = 0$ $f(x) = 0$

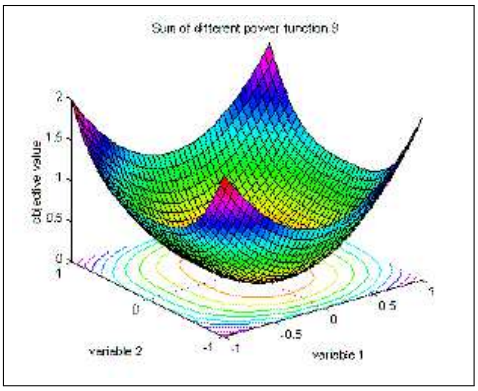

Figura D.14: Representación de la función sum of different Powers para $[-1, 1]$ [Poh06]

Ackley's Path function La función Ackley's Path es del tipo multimdal, comunmente usada.

$$f_{10}(x_i) = -a \cdot e^{-b \cdot \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}} - e^{\frac{\cdot \sum_{i=1}^{n} \cos(c \cdot x_i)}{n}} + a + e^1; \qquad x_i \in [-32{,}768;\ 32{,}768] \tag{D.13}$$

donde: $a = 20,$ $b = 0{,}2,$ $c = 2 \cdot \pi$

mínimo global: $x_i = 0$ $f(x) = 0$

En la figura D.15 se muestra la función para rangos de representación diferentes. La gráfica de la izquierda representa la función en un rango de $[-30, 30]$, mientras que la derecha lo hace próximo al mínimo global, proporcionando así un mayor detalle de las propiedades de la función.

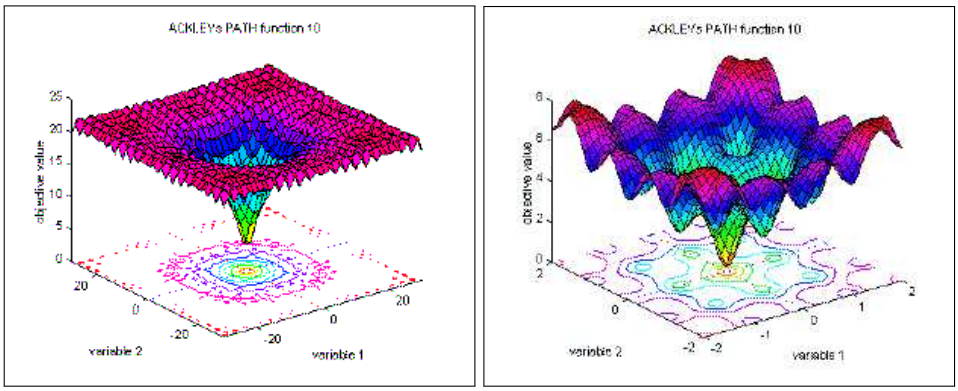

Figura D.15: Representación de la función de Ackley's Path [Poh06]. A la izquierda representada en un área de $([-30, 30])$, a la derecha alrededor del óptimo situado en $[0, 0]$

Langermann's function Función multimodal con mínimos locales irregularmente distribuidos.

$$f_{11}(x_i) = -\sum_{i=1}^{n} c_i \cdot \left(e^{\frac{-\|x_i - A(i)\|^2}{\pi}} \cdot \cos\left(\pi \cdot \|x_i - A(i)\|^2\right) \right); \quad i = 1:n, \quad 2 \leq n \leq 10; \qquad x_i \in [0;\ 10] \tag{D.14}$$

donde:

el valor de A y c pueden ser consultados en el mfile del software

mínimo global: para $n = 5$ $f(x) = -1{,}4$

La figura D.16 muestra la función de Langerman con diferentes variables. La gráfica de la derecha muestra una representación con la primera y la segunda variable. La de la derecha usa la segunda y la tercera variable, quedando la primera con valor 0.

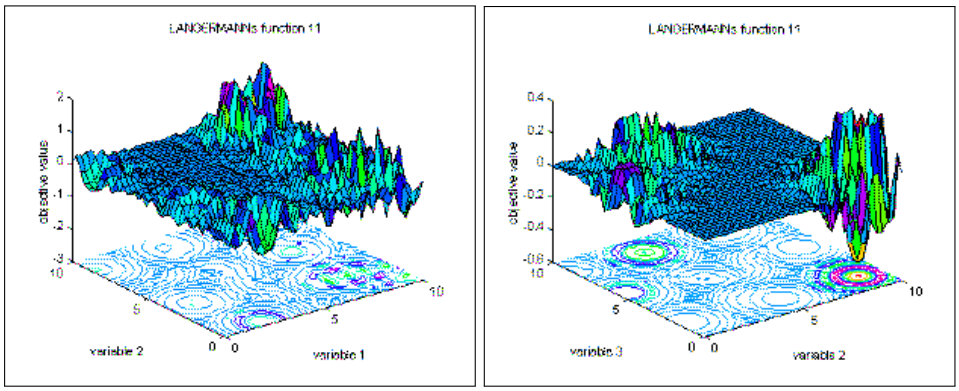

Figura D.16: Representación de la función de Langermann con diferentes variables [Poh06]. En el gráfico de la izquierda en un área de ($[0, 10]$) se representan las variables 1 y 2, en el de la derecha la 2 y la 3 $[0, 0]$

Michalewicz's function La función de Michalewicz es una función multimodal con n! mínimos locales. El parámetro *"m"* define lo "abrupto" de los valles y desfiladeros. Un valor mayor de *m* se traduce en una mayor dificultad de búsqueda. Para valores muy altos de *m* la búsqueda del óptimo se complica de gran modo, ya que los valores de la función fuera de las proximidades de los picos, proporcionan poca información sobre la localización del óptimo ("es como buscar una aguja en un pajar").

$$f_{12}(x_i) = -\sum_{i=1}^{n} sin(x_i) \cdot \left(\sin\left(\frac{i \cdot x(i)^2}{\pi} \right) \right)^{2 \cdot m}; \qquad x_i \in [0;\ \pi] \tag{D.15}$$

donde: $m = 10$

mínimo global: $f(x) = -4{,}687\ \ (n = 5)\,,$
$f(x) = -9{,}66\ \ (n = 10)$

Las dos primeras gráficas (figura D.17) representan una visión global y local de la función de Michalewicz, ambas para dos primeras variables. La tercera gráfica (abajo) representa la función usando la tercera y cuarta variable. Por lo tanto, se puede ver que con el aumento de la dimensión se aumenta la dificultad ya que se introduce un mayor número de valles.

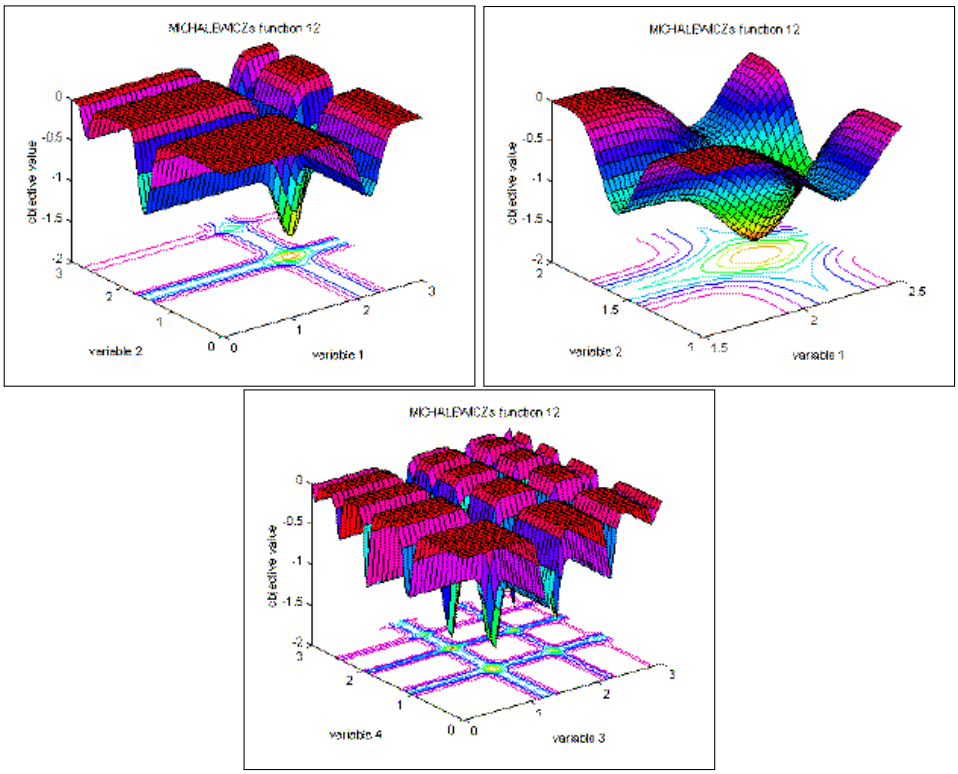

Figura D.17: Representación de la función de Michalewicz [Poh06]. Arriba a la izquierda en un área de $([0, 3])$ para la primera y la segunda variable y a la derecha alrededor del óptimo. Abajo, de igual modo que para arriba a la izquierda en $([0, 3])$ solamente que para la tercera y cuarta variable, con la primera y la segunda a 0

Branins's rcos function Se trata de una función de optimización con 3 óptimos globales.

$$f_{13}(x_1, x_2) = a \cdot \left(x_2 - b \cdot x_1^2 + c \cdot x_1 - d\right)^2 + e \cdot (1 - f) \cdot \cos(x_1) + e; \\ x_1 \in [-5;\ 10], \qquad x_2 \in [0;\ 15] \tag{D.16}$$

donde: $a = 1,\ b = \frac{5{,}1}{4 \cdot \pi^2},\ c = \frac{5}{\pi},\ d = 6,\ e = 10,\ f = \frac{1}{8 \cdot \pi}$

mínimo global: $f(x_1, x_2) = 0{,}397887;$

$(x_1, x_2) = (-\pi, 12{,}275), (\pi, 2{,}275), (9{,}42478, 2{,}475)$

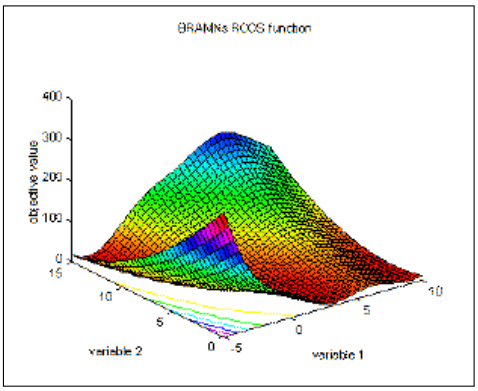

Figura D.18: Representación de la función Branins's rcos para el rango definido

Easom's function La función de Easom es una función unimodal, donde el óptimo global tiene un pequeño área relativo al espacio de búsqueda. Esta función, normalmente de maximización, ha sido invertida para minimización.

$$\begin{aligned} f_{14}(x_1, x_2) &= -\cos(x_1) \cdot cos(x_2) \cdot e^{-\left((x_1-\pi)^2+(x_2-\pi)^2\right)}; \\ & \quad x_1, x_2 \in [-100;\ 100] \end{aligned} \tag{D.17}$$

$$\text{mínimo global:} \qquad (x_1, x_2) = (\pi, \pi); \qquad f(x_1, x_2) = -1$$

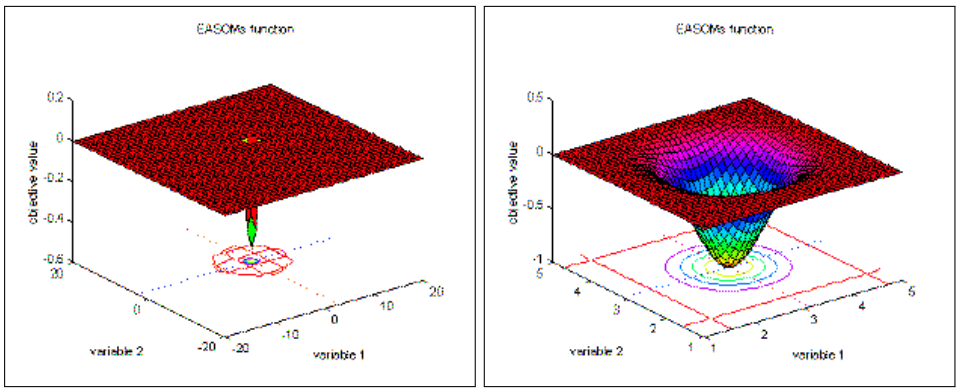

Figura D.19: Representación de la función de Easom [Poh06]; en el gráfico de la izquierda en un área de $([-20, 20])$ y en el de la derecha alrededor del óptimo

Goldstein-Price's function La función Goldstein-Price es una conocida función muy usada para probar la eficacia de métodos de optimización.

$$\begin{aligned} f_{15}(x_1, x_2) &= [1 + (x_1 + x_2 + 1)^2 \cdot \\ &\quad (19 - 14 \cdot x_1 + 3 \cdot {x_1}^2 - 14 \cdot x_2 + 6 \cdot x_1 \cdot x_2 + 3 \cdot {x_2}^2)] \cdot \\ &\quad [30 + (2 \cdot x_1 - 3 \cdot x_2)^2 \cdot \\ &\quad \left(18 - 32 \cdot x_1 + 12 \cdot {x_1}^2 + 48 \cdot x_2 - 36 \cdot x_1 \cdot x_2 + 27 \cdot x_2^2\right)]; \\ &\quad x_1, x_2 \in [-2;\ 2] \end{aligned} \tag{D.18}$$

mínimo global: $(x_1, x_2) = (0, -1), \quad f(x_1, x_2) = 3;$

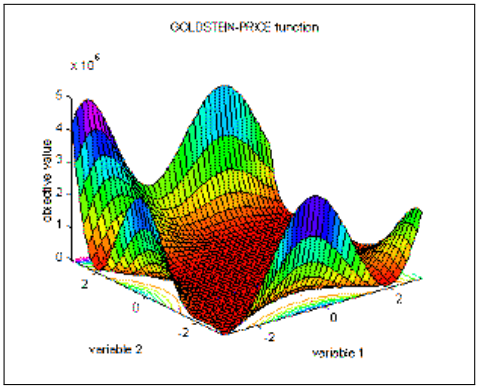

Figura D.20: Representación de la función Goldstein-Price para el rango definido [Poh06]

Six-hump Camel Back function Función de optimización de 2D. En ella, el espacio limitado de búsqueda contiene seis mínimos locales, dos de ellos globales.

$$\begin{aligned} f_{16}(x_1, x_2) &= \left(4 - 2{,}1 \cdot x_1^2 + x_1^{4/3}\right) \cdot x_1^2 + x_1 \cdot x_2 + \left(-4 + 4 \cdot x_2^2\right) \cdot x_2^2; \\ &\quad x_1 \in [-3;\ 3], \quad x_2 \in [-2;\ 2] \end{aligned} \tag{D.19}$$

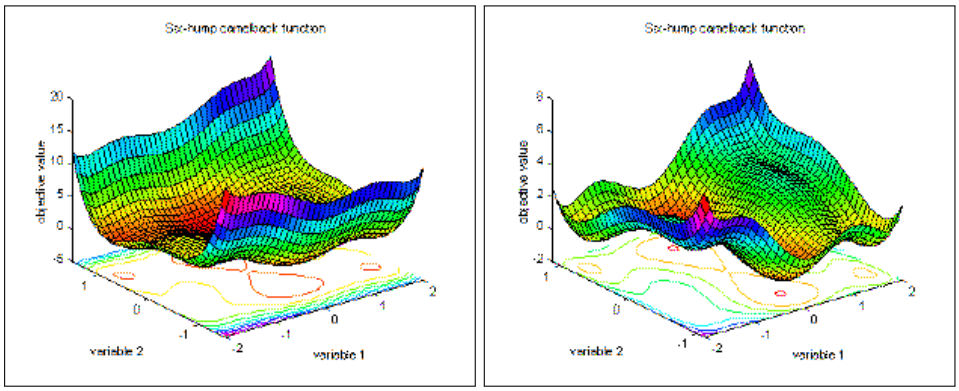

Figura D.21: Representación 3D de la función Six-hump Camel Back [Poh06]. En el gráfico de la izquierda en un área alrededor del mínimo, en el de la derecha lo mismo pero aumentado

D.3.3. Problem Definitions and Evaluation Criteria for the CEC 2005 (PDEC-05)

Como ya se ha comentado en el apartado 3.2.1 la test suite PDEC-05 [SHL+05], fue creada para el CEC 2005 (IEEE Conference on E-Commerce Technology) con el propósito de chequear una test suite estándar, proponiendo a investigadores que trabajaran con ella en la resolución de problemas de optimización, para mostrar los resultados en la mencionada conferencia. Este procedimiento, a diferencia de los mostrados, no solo dispone de una batería de funciones test sino que también ofrece conjuntamente un **procedimiento de evaluación**. Aunque este documento (reporte técnico) está disponible en internet[2], a continuación se incluye el original.

[2]http://www3.ntu.edu.sg/home/EPNSugan/

Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization

P. N. Suganthan[1], N. Hansen[2], J. J. Liang[1], K. Deb[3], Y. -P. Chen[4], A. Auger[2], S. Tiwari[3]

[1]School of EEE, Nanyang Technological University, Singapore, 639798
[2](ETH) Z¨urich, Switzerland
[3]Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology, Kanpur, PIN 208 016, India
[4]Natural Computing Laboratory, Department of Computer Science, National Chiao Tung University, Taiwan

epnsugan@ntu.edu.sg, Nikolaus.Hansen@inf.ethz.ch, liangjing@pmail.ntu.edu.sg, deb@iitk.ac.in, ypchen@csie.nctu.edu.tw, Anne.Auger@inf.ethz.ch, tiwaris@iitk.ac.in

Technical Report, Nanyang Technological University, Singapore
And
KanGAL Report Number 2005005 (Kanpur Genetic Algorithms Laboratory, IIT Kanpur)

May 2005

Acknowledgement: We also acknowledge the contributions by Drs / Professors Maurice Clerc (Maurice.Clerc@WriteMe.com), Bogdan Filipic (bogdan.filipic@ijs.si), William Hart (wehart@sandia.gov), Marc Schoenauer (Marc.Schoenauer@lri.fr), Hans-Paul Schwefel (hans-paul.schwefel@cs.uni-dortmund.de), Aristin Pedro Ballester (p.ballester@imperial.ac.uk) and Darrell Whitley (whitley@CS.ColoState.EDU) .

Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization

In the past two decades, different kinds of optimization algorithms have been designed and applied to solve real-parameter function optimization problems. Some of the popular approaches are real-parameter EAs, evolution strategies (ES), differential evolution (DE), particle swarm optimization (PSO), evolutionary programming (EP), classical methods such as quasi-Newton method (QN), hybrid evolutionary-classical methods, other non-evolutionary methods such as simulated annealing (SA), tabu search (TS) and others. Under each category, there exist many different methods varying in their operators and working principles, such as correlated ES and CMA-ES. In most such studies, a subset of the standard test problems (Sphere, Schwefel's, Rosenbrock's, Rastrigin's, etc.) is considered. Although some comparisons are made in some research studies, often they are confusing and limited to the test problems used in the study. In some occasions, the test problem and chosen algorithm are complementary to each other and the same algorithm may not work in other problems that well. There is definitely a need of evaluating these methods in a more systematic manner by specifying a common termination criterion, size of problems, initialization scheme, linkages/rotation, etc. There is also a need to perform a scalability study demonstrating how the running time/evaluations increase with an increase in the problem size. We would also like to include some real world problems in our standard test suite with codes/executables.

In this report, 25 benchmark functions are given and experiments are conducted on some real-parameter optimization algorithms. The codes in Matlab, C and Java for them could be found at http://www.ntu.edu.sg/home/EPNSugan/. The mathematical formulas and properties of these functions are described in Section 2. In Section 3, the evaluation criteria are given. Some notes are given in Section 4.

1. Summary of the 25 CEC'05 Test Functions

- **Unimodal Functions (5):**
  - F_1: Shifted Sphere Function
  - F_2: Shifted Schwefel's Problem 1.2
  - F_3: Shifted Rotated High Conditioned Elliptic Function
  - F_4: Shifted Schwefel's Problem 1.2 with Noise in Fitness
  - F_5: Schwefel's Problem 2.6 with Global Optimum on Bounds

- **Multimodal Functions (20):**
  - **Basic Functions** (7):
    - F_6: Shifted Rosenbrock's Function
    - F_7: Shifted Rotated Griewank's Function without Bounds
    - F_8: Shifted Rotated Ackley's Function with Global Optimum on Bounds
    - F_9: Shifted Rastrigin's Function
    - F_{10}: Shifted Rotated Rastrigin's Function
    - F_{11}: Shifted Rotated Weierstrass Function
    - F_{12}: Schwefel's Problem 2.13
  - **Expanded Functions** (2):

- ✧ F_{13}: Expanded Extended Griewank's plus Rosenbrock's Function (F8F2)
- ✧ F_{14}: Shifted Rotated Expanded Scaffer's F6

➢ **Hybrid Composition Functions** (11):

- ✧ F_{15}: Hybrid Composition Function
- ✧ F_{16}: Rotated Hybrid Composition Function
- ✧ F_{17}: Rotated Hybrid Composition Function with Noise in Fitness
- ✧ F_{18}: Rotated Hybrid Composition Function
- ✧ F_{19}: Rotated Hybrid Composition Function with a Narrow Basin for the Global Optimum
- ✧ F_{20}: Rotated Hybrid Composition Function with the Global Optimum on the Bounds
- ✧ F_{21}: Rotated Hybrid Composition Function
- ✧ F_{22}: Rotated Hybrid Composition Function with High Condition Number Matrix
- ✧ F_{23}: Non-Continuous Rotated Hybrid Composition Function
- ✧ F_{24}: Rotated Hybrid Composition Function
- ✧ F_{25}: Rotated Hybrid Composition Function without Bounds

➢ **Pseudo-Real Problems:** Available from `http://www.cs.colostate.edu/~genitor/functions.html`. If you have any queries on these problems, please contact Professor Darrell Whitley. Email: whitley@CS.ColoState.EDU

2. Definitions of the 25 CEC'05 Test Functions

2.1 Unimodal Functions:

2.1.1. F_1: *Shifted Sphere Function*

$$F_1(\mathbf{x}) = \sum_{i=1}^{D} z_i^2 + f\_bias_1, \mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions. $\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum.

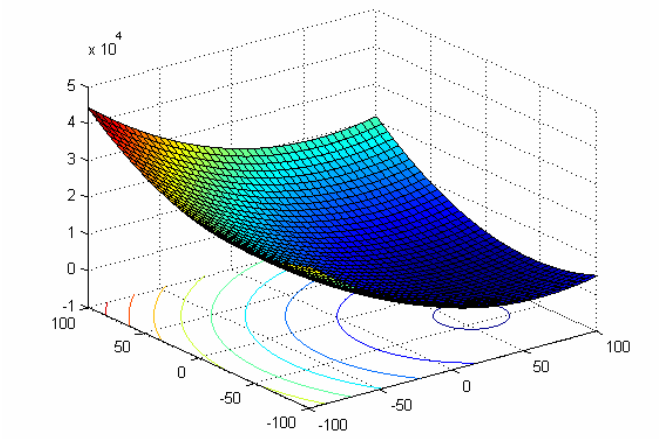

Figure 2-1 3-*D* map for 2-*D* function

Properties:

- Unimodal
- Shifted
- Separable
- Scalable
- $\mathbf{x} \in [-100,100]^D$, Global optimum: $\mathbf{x}^* = \mathbf{o}$, $F_1(\mathbf{x}^*) = f\_bias_1 = -450$

Associated Data files:

Name: sphere_func_data.mat
sphere_func_data.txt
Variable: **o** 1*100 vector the shifted global optimum
When using, cut **o**=**o**(1:*D*)

Name: fbias_data.mat
fbias_data.txt
Variable: **f_bias** 1*25 vector, record all the 25 function's f_bias_i

2.1.2. F_2: *Shifted Schwefel's Problem 1.2*

$$F_2(\mathbf{x}) = \sum_{i=1}^{D} (\sum_{j=1}^{i} z_j)^2 + f\_bias_2, \ \mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

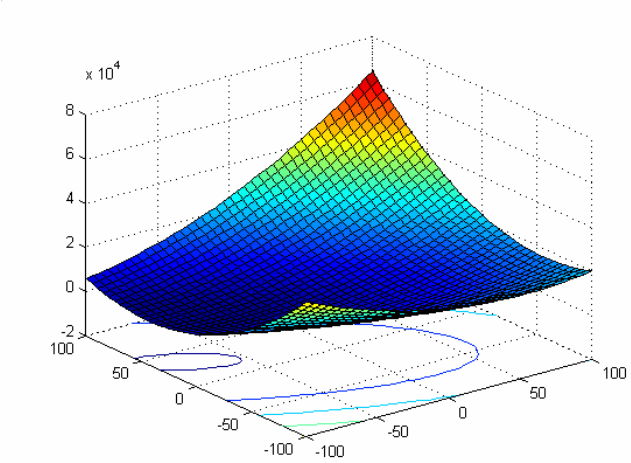

Figure 2-2 3-*D* map for 2-*D* function

Properties:

- Unimodal
- Shifted
- Non-separable
- Scalable
- $\mathbf{x} \in [-100,100]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}, F_2(\mathbf{x}^*) = f\_bias_2 = -450$

Associated Data files:

Name: schwefel_102_data.mat
schwefel_102_data.txt

Variable: **o** 1*100 vector the shifted global optimum
When using, cut **o**=**o**(1:*D*)

2.1.3. F_3: *Shifted Rotated High Conditioned Elliptic Function*

$$F_3(\mathbf{x}) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} z_i^2 + f\_bias_3,\ \mathbf{z} = (\mathbf{x} - \mathbf{o}) * \mathbf{M},\ \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

M: orthogonal matrix

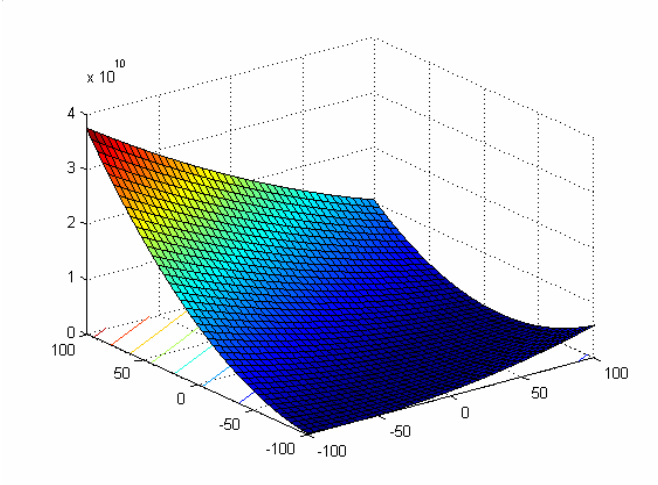

Figure 2-3 3-*D* map for 2-*D* function

Properties:

- Unimodal
- Shifted
- Rotated
- Non-separable
- Scalable
- $\mathbf{x} \in [-100,100]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_3(\mathbf{x}^*) = f\_bias_3$ =- 450

Associated Data files:

Name:	high_cond_elliptic_rot_data.mat high_cond_elliptic_rot_data.txt		
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		
Name:	elliptic_M_D10 .mat		elliptic_M_D10 .txt
Variable:	**M**	10*10 matrix	
Name:	elliptic_M_D30 .mat		elliptic_M_D30 .txt
Variable:	**M**	30*30 matrix	
Name:	elliptic_M_D50 .mat		elliptic_M_D50 .txt
Variable:	**M**	50*50 matrix	

2.1.4. F_4: *Shifted Schwefel's Problem 1.2 with Noise in Fitness*

$$F_4(\mathbf{x}) = (\sum_{i=1}^{D}(\sum_{j=1}^{i} z_j)^2) * (1 + 0.4|N(0,1)|) + f\_bias_4,\ \mathbf{z} = \mathbf{x} - \mathbf{o},\ \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

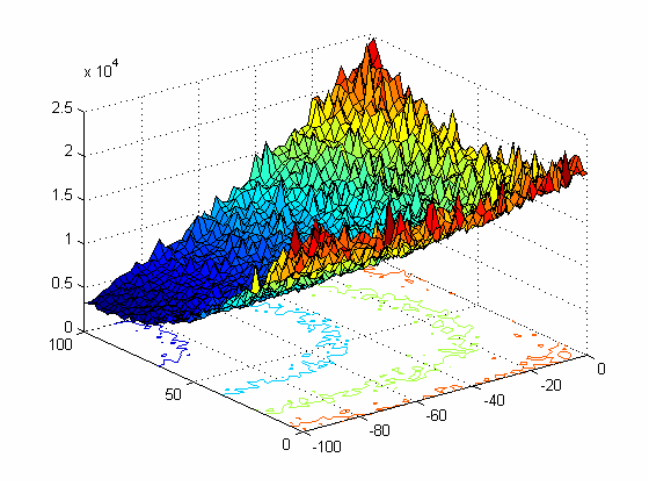

Figure 2-4 3-*D* map for 2-*D* function

Properties:

- Unimodal
- Shifted
- Non-separable
- Scalable
- Noise in fitness
- $\mathbf{x} \in [-100,100]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_4(\mathbf{x}^*) = f\_bias_4 = -450$

Associated Data file:

Name: schwefel_102_data.mat
schwefel_102_data.txt

Variable: **o** 1*100 vector the shifted global optimum
When using, cut **o**=**o**(1:*D*)

2.1.5. *F_5: Schwefel's Problem 2.6 with Global Optimum on Bounds*

$f(\mathbf{x}) = \max\{|x_1 + 2x_2 - 7|, |2x_1 + x_2 - 5|\}, i = 1,...,n$, $\mathbf{x}^* = [1,3]$, $f(\mathbf{x}^*) = 0$

Extend to *D* dimensions:

$F_5(\mathbf{x}) = \max\{|\mathbf{A}_i\mathbf{x} - \mathbf{B}_i|\} + f\_bias_5, i = 1,...,D, \mathbf{x} = [x_1, x_2, ..., x_D]$

D: dimensions

A is a *D***D* matrix, a_{ij} are integer random numbers in the range [-*500, 500*], $\det(\mathbf{A}) \neq 0$, $\mathbf{A}_i$ is the i^{th} row of **A**.

$\mathbf{B}_i = \mathbf{A}_i * \mathbf{o}$, **o** is a *D**1 vector, o_i are random number in the range [-100,100]

After load the data file, set $o_i = -100$, for $i = 1,2,...,\lceil D/4 \rceil$, $o_i = 100$, for $i = \lfloor 3D/4 \rfloor,...,D$

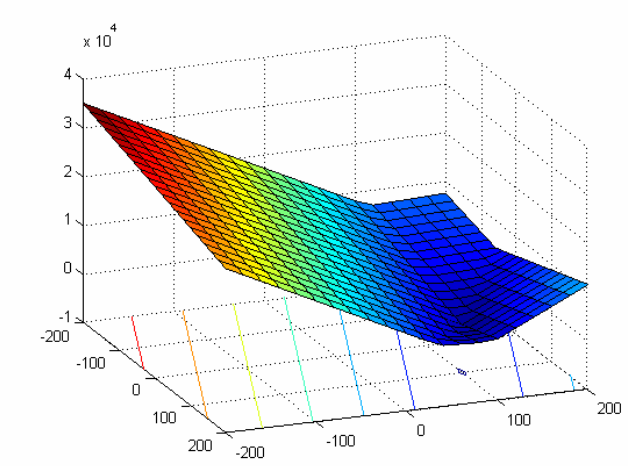

Figure 2-5 3-*D* map for 2-*D* function

Properties:

- Unimodal
- Non-separable
- Scalable
- If the initialization procedure initializes the population at the bounds, this problem will be solved easily.
- $\mathbf{x} \in [-100,100]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_5(\mathbf{x}^*) = f\_bias_5 = -310$

Associated Data file:

Name: schwefel_206_data.mat
schwefel_206_data.txt

Variable: **o** 1*100 vector the shifted global optimum
A 100*100 matrix
When using, cut **o**=**o**(1:*D*) **A**=**A**(1:*D*,1:*D*)
In schwefel_206_data.txt ,the first line is **o** (1*100 vector),and line2-line101 is **A**(100*100 matrix)

2.2 Basic Multimodal Functions

2.2.1. F_6: *Shifted Rosenbrock's Function*

$$F_6(\mathbf{x}) = \sum_{i=1}^{D-1}(100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f\_bias_6, \ \mathbf{z} = \mathbf{x} - \mathbf{o} + 1, \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

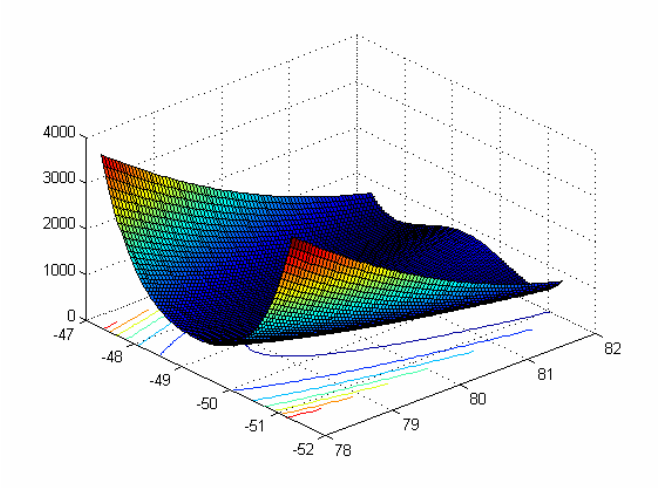

Figure 2-6 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Non-separable
- Scalable
- Having a very narrow valley from local optimum to global optimum
- $\mathbf{x} \in [-100,100]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_6(\mathbf{x}^*) = f\_bias_6 = 390$

Associated Data file:

Name:	rosenbrock_func_data.mat		
	rosenbrock_func_data.txt		
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		

2.2.2. F_7: *Shifted Rotated Griewank's Function without Bounds*

$$F_7(\mathbf{x}) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{z_i}{\sqrt{i}}) + 1 + f\_bias_7 \ , \ \mathbf{z} = (\mathbf{x} - \mathbf{o}) * \mathbf{M} \ , \ \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

M': linear transformation matrix, condition number=3

M =**M'**(1+0.3|N(0,1)|)

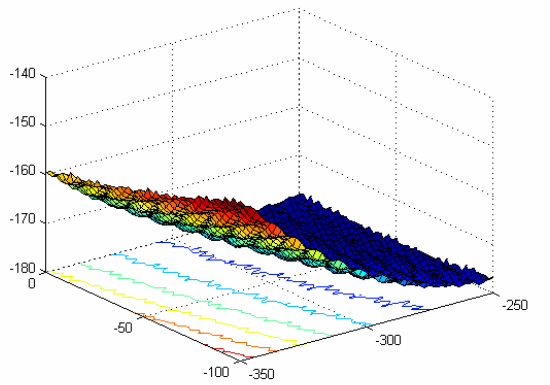

Figure 2-7 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Rotated
- Shifted
- Non-separable
- Scalable
- No bounds for variables *x*
- Initialize population in $[0,600]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$ is outside of the initialization range, $F_7(\mathbf{x}^*) = f\_bias_7 = -180$

Associated Data file:

Name:	griewank_func_data.mat		griewank_func_data.txt
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		
Name:	griewank_M_D10 .mat		griewank_M_D10 .txt
Variable:	**M**	10*10 matrix	
Name:	griewank_M_D30 .mat		griewank_M_D30 .txt
Variable:	**M**	30*30 matrix	
Name:	griewank_M_D50 .mat		griewank_M_D50 .txt
Variable:	**M**	50*50 matrix	

2.2.3. *F_8: Shifted Rotated Ackley's Function with Global Optimum on Bounds*

$$F_8(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)) + 20 + e + f\_bias_8, \ \mathbf{z} = (\mathbf{x} - \mathbf{o}) * \mathbf{M},$$

$\mathbf{x} = [x_1, x_2, ..., x_D]$, *D:* dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum;

After load the data file, set $o_{2j-1} = -32$ o_{2j} are randomly distributed in the search range, for $j = 1, 2, ..., \lfloor D/2 \rfloor$

M: linear transformation matrix, condition number=100

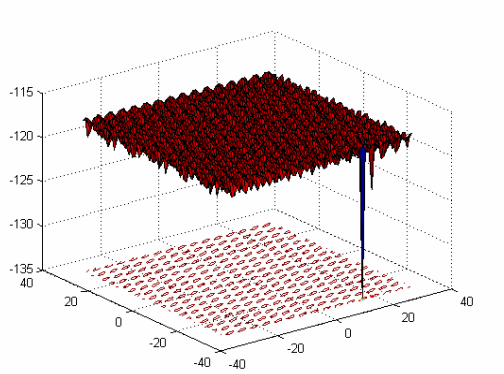

Figure 2-8 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Rotated
- Shifted
- Non-separable
- Scalable
- **A**'s condition number Cond(**A**) increases with the number of variables as $O(D^2)$
- Global optimum on the bound
- If the initialization procedure initializes the population at the bounds, this problem will be solved easily.
- $\mathbf{x} \in [-32, 32]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_8(\mathbf{x}^*) = f\_bias_8 = -140$

Associated Data file:

Name:	ackley_func_data.mat		ackley_func_data.txt
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		
Name:	ackley_M_D10 .mat		ackley_M_D10 .txt
Variable:	**M**	10*10 matrix	
Name:	ackley_M_D30 .mat		ackley_M_D30 .txt
Variable:	**M**	30*30 matrix	
Name:	ackley_M_D50 .mat		ackley_M_D50 .txt
Variable:	**M**	50*50 matrix	

2.2.4. F_9: *Shifted Rastrigin's Function*

$$F_9(\mathbf{x}) = \sum_{i=1}^{D}(z_i^{\,2} - 10\cos(2\pi z_i) + 10) + f\_bias_9,\ \mathbf{z} = \mathbf{x} - \mathbf{o},\ \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\boldsymbol{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

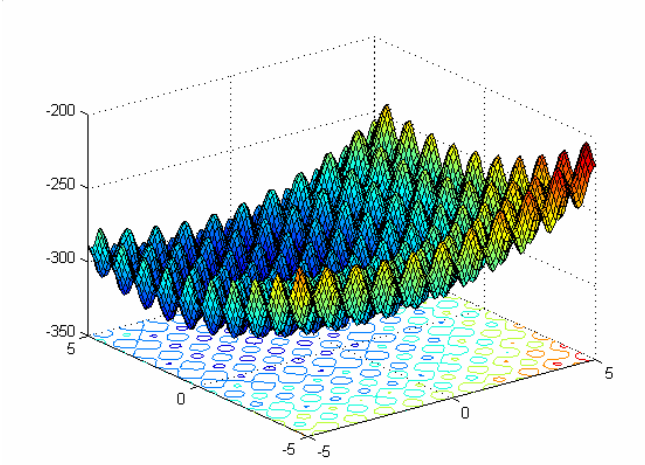

Figure 2-9 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Separable
- Scalable
- Local optima's number is huge
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_9(\boldsymbol{x}^*) = f\_bias_9 = -330$

Associated Data file:

Name:	rastrigin_func_data.mat		
	rastrigin_func_data.txt		
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		

2.2.5. F_{10}: *Shifted Rotated Rastrigin's Function*

$$F_{10}(\mathbf{x}) = \sum_{i=1}^{D}(z_i^{\,2} - 10\cos(2\pi z_i) + 10) + f\_bias_{10},\ \mathbf{z} = (\mathbf{x} - \mathbf{o}) * \mathbf{M}, \mathbf{x} = [x_1, x_2, ..., x_D]$$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

M: linear transformation matrix, condition number=2

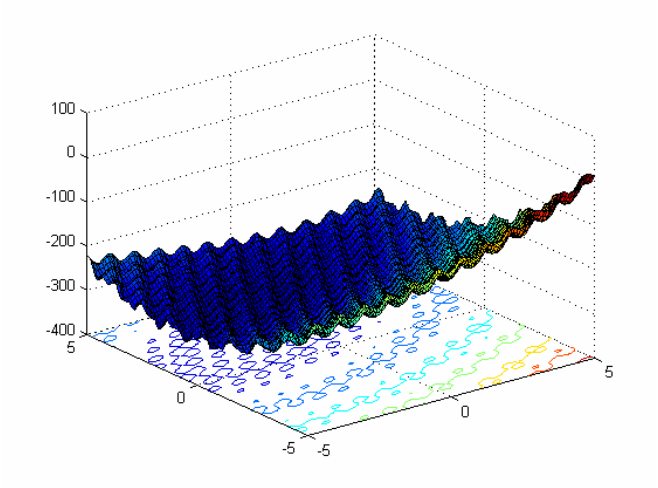

Figure 2-10 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Rotated
- Non-separable
- Scalable
- Local optima's number is huge
- $\mathbf{x} \in [-5, 5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_{10}(\mathbf{x}^*) = f\_bias_{10}$ = - 330

Associated Data file:

Name:	rastrigin_func_data.mat rastrigin_func_data.txt		
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		
Name:	rastrigin_M_D10 .mat		rastrigin_M_D10 .txt
Variable:	**M**	10*10 matrix	
Name:	rastrigin_M_D30 .mat		rastrigin_M_D30 .txt
Variable:	**M**	30*30 matrix	
Name:	rastrigin_M_D50 .mat		rastrigin_M_D50 .txt
Variable:	**M**	50*50 matrix	

2.2.6. F_{11}: *Shifted Rotated Weierstrass Function*

$$F_{11}(\mathbf{x}) = \sum_{i=1}^{D}\left(\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k (z_i + 0.5))]\right) - D\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k \cdot 0.5)] + f\_bias_{11},$$

a=0.5, b=3, k_{max}=20, $\mathbf{z} = (\mathbf{x} - \mathbf{o}) * \mathbf{M}$, $\mathbf{x} = [x_1, x_2, ..., x_D]$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

M: linear transformation matrix, condition number=5

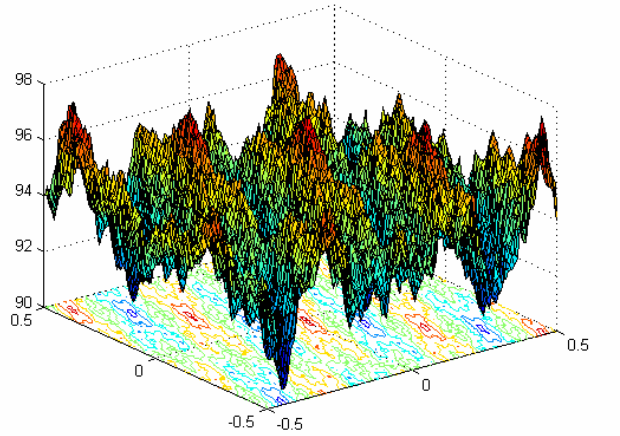

Figure 2-11 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Rotated
- Non-separable
- Scalable
- Continuous but differentiable only on a set of points
- $\mathbf{x} \in [-0.5, 0.5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_{11}(\mathbf{x}^*) = f\_bias_{11} = 90$

Associated Data file:

Name:	weierstrass_data.mat		weierstrass_data.txt
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		
Name:	weierstrass_M_D10 .mat		weierstrass_M_D10 .txt
Variable:	**M**	10*10 matrix	
Name:	weierstrass_M_D30 .mat		weierstrass_M_D30 .txt
Variable:	**M**	30*30 matrix	
Name:	weierstrass_M_D50 .mat		weierstrass_M_D50 .txt
Variable:	**M**	50*50 matrix	

2.2.7. F_{12}: *Schwefel's Problem 2.13*

$$F_{12}(\mathbf{x}) = \sum_{i=1}^{D} (\mathbf{A}_i - \mathbf{B}_i(\boldsymbol{x}))^2 + f\_bias_{12}, \mathbf{x} = [x_1, x_2, ..., x_D]$$

$$\mathbf{A}_i = \sum_{j=1}^{D} (a_{ij} \sin \alpha_j + b_{ij} \cos \alpha_j), \mathbf{B}_i(x) = \sum_{j=1}^{D} (a_{ij} \sin x_j + b_{ij} \cos x_j), \text{ for } i = 1, ..., D$$

D: dimensions

A, **B** are two *D***D* matrix, a_{ij}, b_{ij} are integer random numbers in the range [-100,100],

$\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_D], \alpha_j$ are random numbers in the range $[-\pi, \pi]$.

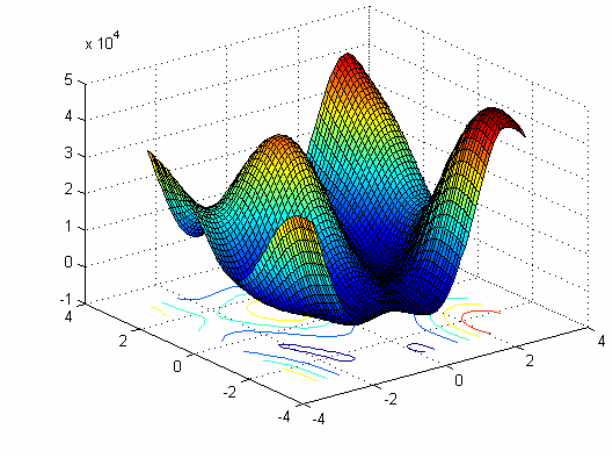

Figure 2-12 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Non-separable
- Scalable
- $\mathbf{x} \in [-\pi, \pi]^D$, Global optimum $\mathbf{x}^* = \boldsymbol{\alpha}$, $F_{12}(\mathbf{x}^*) = f\_bias_{12}$ = - 460

Associated Data file:

Name: schwefel_213_data.mat
schwefel_213_data.txt

Variable: **alpha** 1*100 vector the shifted global optimum
a 100*100 matrix
b 100*100 matrix
When using, cut **alpha**=**alpha**(1:*D*) **a**=**a**(1:*D*,1:*D*) **b**=**b**(1:*D*,1:*D*)
In schwefel_213_data.txt, and line1-line100 is **a** (100*100 matrix),and line101-line200 is **b** (100*100 matrix), the last line is **alpha**(1*100 vector),

2.3 Expanded Functions

Using a 2-*D* function $F(x, y)$ as a starting function, corresponding expanded function is:

$$EF(x_1, x_2, ..., x_D) = F(x_1, x_2) + F(x_2, x_3) + ... + F(x_{D-1}, x_D) + F(x_D, x_1)$$

2.3.1. F_{13}: *Shifted Expanded Griewank's plus Rosenbrock's Function (F8F2)*

F8: Griewank's Function: $F8(\boldsymbol{x}) = \sum_{i=1}^{D} \frac{x_i^{\,2}}{4000} - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$

F2: Rosenbrock's Function: $F2(\boldsymbol{x}) = \sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$

$$F8F2(x_1, x_2, ..., x_D) = F8(F2(x_1, x_2)) + F8(F2(x_2, x_3)) + ... + F8(F2(x_{D-1}, x_D)) + F8(F2(x_D, x_1))$$

Shift to

$$F_{13}(\mathbf{x}) = F8(F2(z_1, z_2)) + F8(F2(z_2, z_3)) + ... + F8(F2(z_{D-1}, z_D)) + F8(F2(z_D, z_1)) + f\_bias_{13}$$

$\mathbf{z} = \mathbf{x} - \mathbf{o} + 1$, $\mathbf{x} = [x_1, x_2, ..., x_D]$

D: dimensions $\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

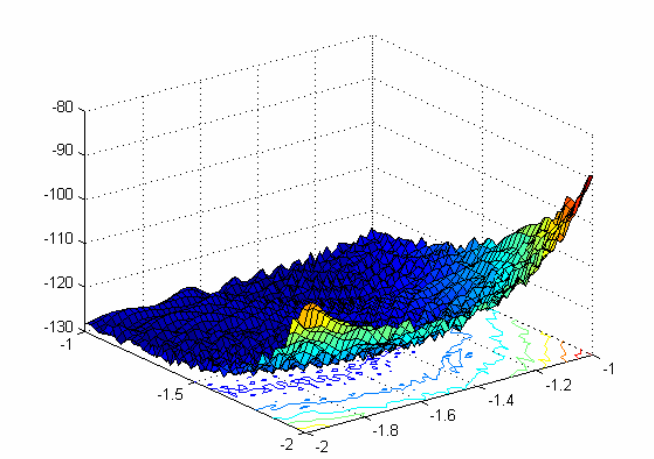

Figure 2-13 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Non-separable
- Scalable
- $\mathbf{x} \in [-3,1]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_{13}(\mathbf{x}^*) = f\_bias_{13}(13)$=-130

Associated Data file:

Name: EF8F2_func_data.mat
EF8F2_func_data.txt

Variable: **o** 1*100 vector the shifted global optimum
When using, cut **o**=**o**(1:*D*)

2.3.2. F_{14}: *Shifted Rotated Expanded Scaffer's F6 Function*

$$F(x,y) = 0.5 + \frac{(\sin^2(\sqrt{x^2+y^2}) - 0.5)}{(1+0.001(x^2+y^2))^2}$$

Expanded to

$F_{14}(\mathbf{x}) = EF(z_1, z_2, ..., z_D) = F(z_1, z_2) + F(z_2, z_3) + ... + F(z_{D-1}, z_D) + F(z_D, z_1) + f\_bias_{14}$,

$\mathbf{z} = (\mathbf{x} - \mathbf{o}) * \mathbf{M}$, $\mathbf{x} = [x_1, x_2, ..., x_D]$

D: dimensions

$\mathbf{o} = [o_1, o_2, ..., o_D]$: the shifted global optimum

M: linear transformation matrix, condition number=3

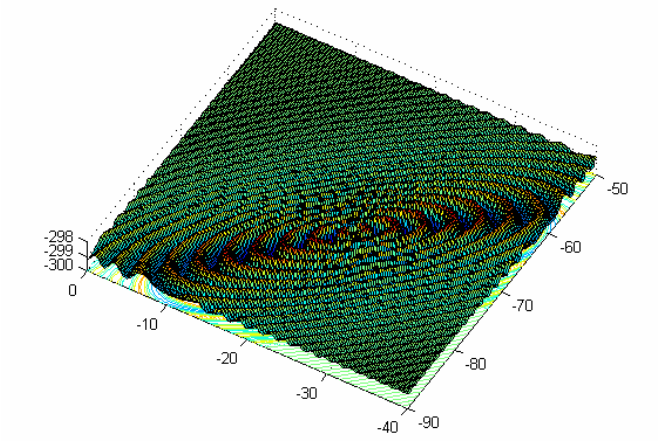

Figure 2-14 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Shifted
- Non-separable
- Scalable
- $\mathbf{x} \in [-100,100]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}$, $F_{14}(\mathbf{x}^*) = f\_bias_{14}$ (14)= -300

Associated Data file:

Name:	E_ScafferF6_func_data.mat		E_ScafferF6_func_data.txt
Variable:	**o**	1*100 vector	the shifted global optimum
	When using, cut **o**=**o**(1:*D*)		
Name:	E_ScafferF6_M_D10 .mat		E_ScafferF6_M_D10 .txt
Variable:	**M**	10*10 matrix	
Name:	E_ScafferF6_M_D30 .mat		E_ScafferF6_M_D30 .txt
Variable:	**M**	30*30 matrix	
Name:	E_ScafferF6_M_D50 .mat		E_ScafferF6_M_D50 .txt
Variable:	**M**	50*50 matrix	

2.4 Composition functions

$F(\mathbf{x})$: new composition function
$f_i(\mathbf{x})$: i[th] basic function used to construct the composition function
n: number of basic functions
D: dimensions
$\mathbf{M}_i$: linear transformation matrix for each $f_i(\boldsymbol{x})$
$\mathbf{o}_i$: new shifted optimum position for each $f_i(\boldsymbol{x})$

$$F(\mathbf{x}) = \sum_{i=1}^{n} \{w_i * [f_i'((\mathbf{x} - \mathbf{o}_i)/\lambda_i * \mathbf{M}_i) + bias_i]\} + f\_bias$$

w_i: weight value for each $f_i(\mathbf{x})$, calculated as below:

$$w_i = \exp\left(-\frac{\sum_{k=1}^{D}(x_k - o_{ik})^2}{2D\sigma_i^2}\right),$$

$$w_i = \begin{cases} w_i & w_i == \max(w_i) \\ w_i * (1-\max(w_i).\hat{}10) & w_i \neq \max(w_i) \end{cases}$$

$$\text{then normalize the weight } w_i = w_i / \sum_{i=1}^{n} w_i$$

σ_i: used to control each $f_i(\mathbf{x})$'s coverage range, a small σ_i give a narrow range for that $f_i(\mathbf{x})$
λ_i: used to stretch compress the function, λ_i>1 means stretch, λ_i<1 means compress
$\mathbf{o}_i$ define the global and local optima's position, $bias_i$ define which optimum is global optimum. Using $\mathbf{o}_i$, $bias_i$, a global optimum can be placed anywhere.

If $f_i(\mathbf{x})$ are different functions, different functions have different properties and height, in order to get a better mixture, estimate a biggest function value $f_{\max i}$ for 10 functions $f_i(\mathbf{x})$, then normalize each basic functions to similar heights as below:
$f_i'(\mathbf{x}) = C * f_i(\mathbf{x}) / |f_{\max i}|$, C is a predefined constant.
$|f_{\max i}|$ is estimated using $|f_{\max i}| = f_i((\mathbf{x}'/\lambda_i) * \mathbf{M}_i)$, $\mathbf{x}'$=[5,5…,5].

In the following composition functions,
Number of basic functions n=10.
D: dimensions
$\mathbf{o}$: n*D matrix, defines $f_i(\mathbf{x})$'s global optimal positions
bias=[0, 100, 200, 300, 400, 500, 600, 700, 800, 900]. Hence, the first function $f_1(\mathbf{x})$ always the function with the global optimum.
C=2000

Pseudo Code:

Define f1-f10, σ, λ, bias, C, load data file o and rotated linear transformation matrix **M1-M10**
$\mathbf{y}$ =[5,5…,5].
For i=1:10

$$w_i = \exp(-\frac{\sum_{k=1}^{D}(x_k - o_{ik})^2}{2D\sigma_i^2}),$$

$$fit_i = f_i(((\mathbf{x} - \mathbf{o}_i)/\lambda_i)*\mathbf{M}_i)$$

$$f\max_i = f_i((\mathbf{y}/\lambda_i)*\mathbf{M}_i),$$

$$fit_i = C*fit_i / f\max_i$$

EndFor

$$SW = \sum_{i=1}^{n} w_i$$

$$MaxW = \max(w_i)$$

For i=1:10

$$w_i = \begin{cases} w_i & if \quad w_i == MaxW \\ w_i*(1\text{-}MaxW.\text{^}10) & if \quad w_i \neq MaxW \end{cases}$$

$$w_i = w_i / SW$$

EndFor

$$F(\mathbf{x}) = \sum_{i=1}^{n}\{w_i*[fit_i + bias_i]\}$$

$$F(\mathbf{x}) = F(\boldsymbol{x}) + f\_bias$$

2.4.1. *F_{15}: Hybrid Composition Function*

$f_{1-2}(\mathbf{x})$: Rastrigin's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$$

$f_{3-4}(\mathbf{x})$: Weierstrass Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k (x_i + 0.5))]) - D\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k \cdot 0.5)],$$

a=0.5, b=3, k_{max}=20

$f_{5-6}(\mathbf{x})$: Griewank's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1$$

$f_{7-8}(\mathbf{x})$: Ackley's Function

$$f_i(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)) + 20 + e$$

$f_{9-10}(\mathbf{x})$: Sphere Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}x_i^2$$

$\sigma_i = 1$ for $i = 1, 2, ..., D$

$\boldsymbol{\lambda}$ = [1, 1, 10, 10, 5/60, 5/60, 5/32, 5/32, 5/100, 5/100]

$\mathbf{M}_i$ are all identity matrices

Please notice that these formulas are just for the basic functions, no shift or rotation is included in these expressions. x here is just a variable in a function.

Take f_1 as an example, when we calculate $f_1(((\mathbf{x} - \mathbf{o}_1)/\lambda_1)*\mathbf{M}_1)$, we need calculate $f_1(\mathbf{z}) = \sum_{i=1}^{D}(z_i^2 - 10\cos(2\pi z_i) + 10)$, $\mathbf{z} = ((\mathbf{x} - \mathbf{o}_1)/\lambda_1)*\mathbf{M}_1$.

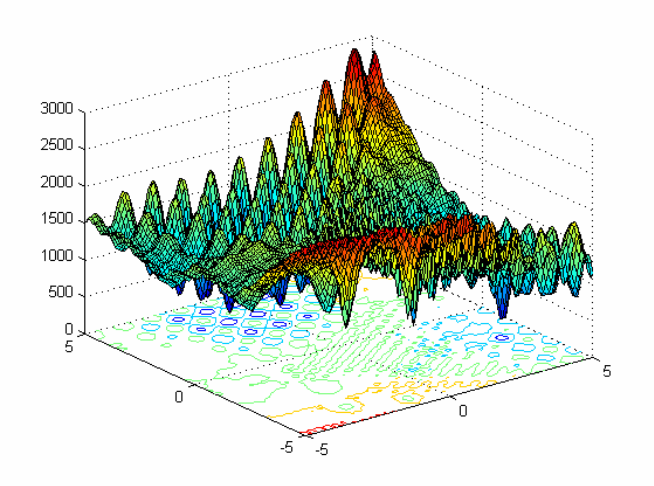

Figure 2-15 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Separable near the global optimum (Rastrigin)
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Sphere Functions give two flat areas for the function
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{15}(\mathbf{x}^*) = f\_bias_{15} = 120$

Associated Data file:

Name: hybrid_func1_data.mat
hybrid_func1_data.txt

Variable: **o** 10*100 vector the shifted optimum for 10 functions
When using, cut **o**=**o**(:,1:*D*)

2.4.2. F_{16}: *Rotated Version of Hybrid Composition Function* F_{15}

Except $\mathbf{M}_i$ are different linear transformation matrixes with condition number of 2, all other settings are the same as F_{15}.

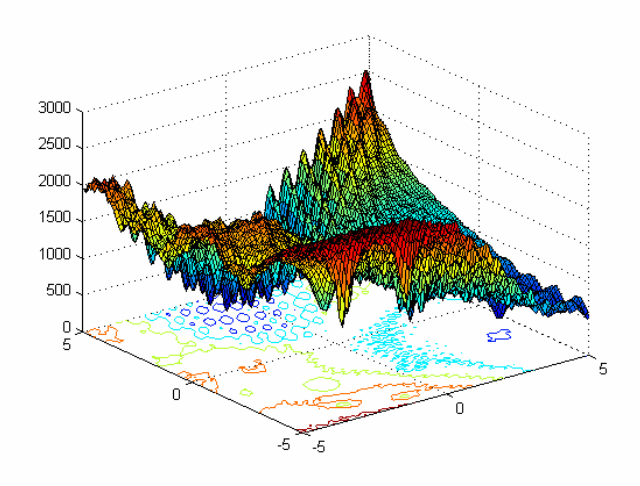

Figure 2-16 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Rotated
- Non-Separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Sphere Functions give two flat areas for the function.
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{16}(\boldsymbol{x}^*) = f\_bias_{16} = 120$

Associated Data file:

Name: hybrid_func1_data.mat
hybrid_func1_data.txt
Variable: **o** 10*100 vector the shifted optima for 10 functions
When using, cut **o**=**o**(:,1:*D*)

Name: hybrid_func1_M_D10 .mat
Variable: **M** an structure variable
Contains **M.M1 M.M2**, … , **M.M10** ten 10*10 matrixes
Name: hybrid_func1_M_D10 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 10*10 matrixes, 1-10 lines are **M1**, 11-20 lines are **M2**,....,91-100 lines are **M10**

Name: hybrid_func1_M_D30 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 30*30 matrix
Name: hybrid_func1_M_D30 .txt

Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 30*30 matrixes, 1-30 lines are **M1**, 31-60 lines are **M2**,....,271-300 lines are **M10**

Name: hybrid_func1_M_D50 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 50*50 matrix
Name: hybrid_func1_M_D50 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 50*50 matrixes, 1-50 lines are **M1**, 51-100 lines are **M2**,....,451-500 lines are **M10**

2.4.3. *F_{17}: F_{16} with Noise in Fitness*

Let (F_{16} - *f_bias*$_{16}$) be $G(\boldsymbol{x})$, then

$F_{17}(\mathbf{x}) = G(\mathbf{x})*(1+0.2|\mathrm{N}(0,1)|) + f\_bias_{17}$

All settings are the same as F_{16}.

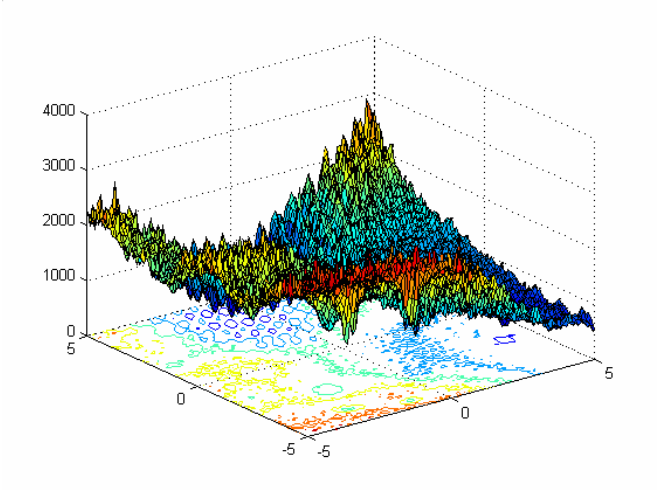

Figure 2-17 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Rotated
- Non-Separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Sphere Functions give two flat areas for the function.
- With Gaussian noise in fitness
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{17}(\mathbf{x}^*) = f\_bias_{17}$ =120

Associated Data file:

Same as F_{16}.

2.4.4. F_{18}: *Rotated Hybrid Composition Function*

$f_{1-2}(\mathbf{x})$: Ackley's Function

$$f_i(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)) + 20 + e$$

$f_{3-4}(\mathbf{x})$: Rastrigin's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$$

$f_{5-6}(\mathbf{x})$: Sphere Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$$

$f_{7-8}(\mathbf{x})$: Weierstrass Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k (x_i + 0.5))]) - D\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k \cdot 0.5)],$$

a=0.5, b=3, k_{max}=20

$f_{9-10}(\mathbf{x})$: Griewank's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1$$

$\boldsymbol{\sigma}$ =[1, 2, 1.5, 1.5, 1, 1, 1.5, 1.5, 2, 2];

$\boldsymbol{\lambda}$ = [2*5/32; 5/32; 2*1; 1; 2*5/100; 5/100; 2*10; 10; 2*5/60; 5/60]

$\mathbf{M}_i$ are all rotation matrices. Condition numbers are [2 3 2 3 2 3 20 30 200 300]

$\mathbf{o}_{10} = [0, 0, ..., 0]$

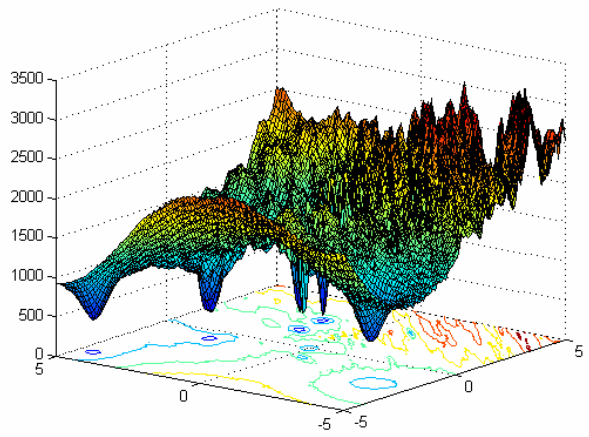

Figure 2-18 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Rotated
- Non-Separable
- Scalable

- A huge number of local optima
- Different function's properties are mixed together
- Sphere Functions give two flat areas for the function.
- A local optimum is set on the origin
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{18}(\boldsymbol{x}^*) = f\_bias_{18} = 10$

Associated Data file:

Name: hybrid_func2_data.mat
hybrid_func2_data.txt
Variable: **o** 10*100 vector the shifted optima for 10 functions
When using, cut **o**=**o**(:,1:*D*)

Name: hybrid_func2_M_D10 .mat
Variable: **M** an structure variable
Contains M.**M1 M.M2, … , M.M10** ten 10*10 matrixes
Name: hybrid_func2_M_D10 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 10*10 matrixes, 1-10 lines are **M1**, 11-20 lines are **M2**,....,91-100 lines are **M10**

Name: hybrid_func2_M_D30 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 30*30 matrix
Name: hybrid_func2_M_D30 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 30*30 matrixes, 1-30 lines are **M1**, 31-60 lines are **M2**,....,271-300 lines are M10

Name: hybrid_func2_M_D50 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 50*50 matrix
Name: hybrid_func2_M_D50 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 50*50 matrixes, 1-50 lines are **M1**, 51-100 lines are **M2**,....,451-500 lines are **M10**

2.4.5. F_{19}: *Rotated Hybrid Composition Function with narrow basin global optimum*

All settings are the same as F_{18} except

σ =[0.1, 2, 1.5, 1.5, 1, 1, 1.5, 1.5, 2, 2];,

λ = [0.1*5/32; 5/32; 2*1; 1; 2*5/100; 5/100; 2*10; 10; 2*5/60; 5/60]

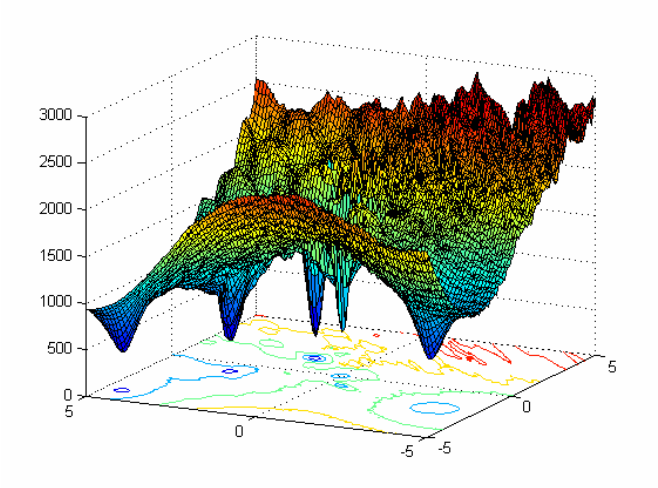

Figure 2-19 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Non-separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Sphere Functions give two flat areas for the function.
- A local optimum is set on the origin
- A narrow basin for the global optimum
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{19}(\mathbf{x}^*) = f\_bias_{19}(19)$=10

Associated Data file:

Same as F_{18}.

2.4.6. F_{20}: *Rotated Hybrid Composition Function with Global Optimum on the Bounds*

All settings are the same as F_{18} except after load the data file, set $o_{1(2j)} = 5$, for $j = 1, 2, ..., \lfloor D/2 \rfloor$

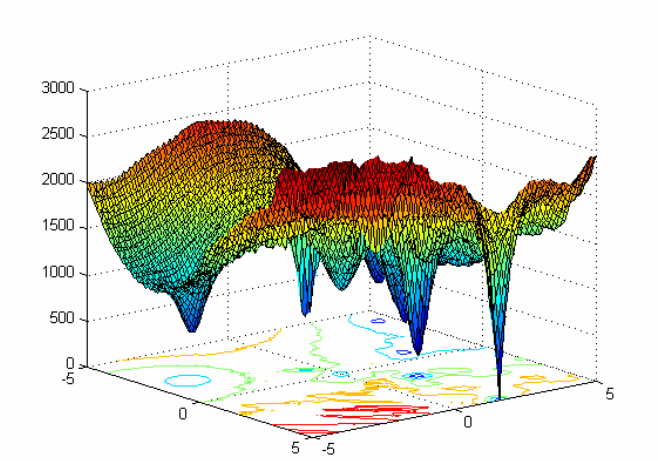

Figure 2-20 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Non-separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Sphere Functions give two flat areas for the function.
- A local optimum is set on the origin
- Global optimum is on the bound
- If the initialization procedure initializes the population at the bounds, this problem will be solved easily.
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{20}(\mathbf{x}^*) = f\_bias_{20}$ =10

Associated Data file:

Same as F_{18}.

2.4.7. F_{21}: *Rotated Hybrid Composition Function*

$f_{1-2}(\mathbf{x})$: Rotated Expanded Scaffer's F6 Function

$$F(x,y) = 0.5 + \frac{(\sin^2(\sqrt{x^2+y^2}) - 0.5)}{(1+0.001(x^2+y^2))^2}$$

$$f_i(\mathbf{x}) = F(x_1,x_2) + F(x_2,x_3) + \ldots + F(x_{D-1},x_D) + F(x_D,x_1)$$

$f_{3-4}(\mathbf{x})$: Rastrigin's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10)$$

$f_{5-6}(\mathbf{x})$: F8F2 Function

$$F8(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$$

$$F2(\mathbf{x}) = \sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$$

$$f_i(\mathbf{x}) = F8(F2(x_1,x_2)) + F8(F2(x_2,x_3)) + \ldots + F8(F2(x_{D-1},x_D)) + F8(F2(x_D,x_1))$$

$f_{7-8}(\mathbf{x})$: Weierstrass Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D} (\sum_{k=0}^{k\max} [a^k \cos(2\pi b^k (x_i + 0.5))]) - D\sum_{k=0}^{k\max} [a^k \cos(2\pi b^k \cdot 0.5)],$$

a=0.5, b=3, k_{max}=20

$f_{9-10}(\mathbf{x})$: Griewank's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$$

$\boldsymbol{\sigma} = [1,1,1,1,1,2,2,2,2,2]$,

$\boldsymbol{\lambda}$ = [5*5/100; 5/100; 5*1; 1; 5*1; 1; 5*10; 10; 5*5/200; 5/200];

$\mathbf{M}_i$ are all orthogonal matrix

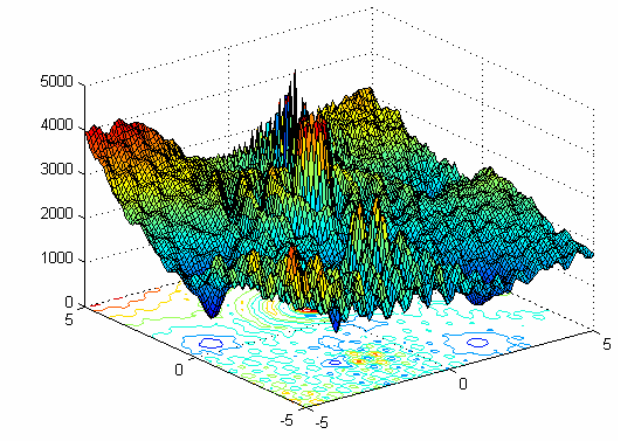

Figure 2-21 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Rotated
- Non-Separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{21}(\mathbf{x}^*) = f\_bias_{21}$ =360

Associated Data file:

Name: hybrid_func3_data.mat
hybrid_func3_data.txt
Variable: **o** 10*100 vector the shifted optima for 10 functions
When using, cut **o**=**o**(:,1:*D*)

Name: hybrid_func3_M_D10 .mat
Variable: **M** an structure variable
Contains **M.M1 M.M2, … , M.M10** ten 10*10 matrixes
Name: hybrid_func3_M_D10 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 10*10 matrixes, 1-10 lines are **M1**, 11-20 lines are **M2,**....,91-100 lines are **M10**

Name: hybrid_func3_M_D30 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 30*30 matrix
Name: hybrid_func3_M_D30 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 30*30 matrixes, 1-30 lines are **M1**, 31-60 lines are **M2**,....,271-300 lines are **M10**

Name: hybrid_func3_M_D50 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 50*50 matrix
Name: hybrid_func3_M_D50 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 50*50 matrixes, 1-50 lines are **M1**, 51-100 lines are **M2**,....,451-500 lines are **M10**

2.4.8. *F_{22}: Rotated Hybrid Composition Function with High Condition Number Matrix*

All settings are the same as F_{21} except $\mathbf{M}_i$'s condition numbers are [10 20 50 100 200 1000 2000 3000 4000 5000]

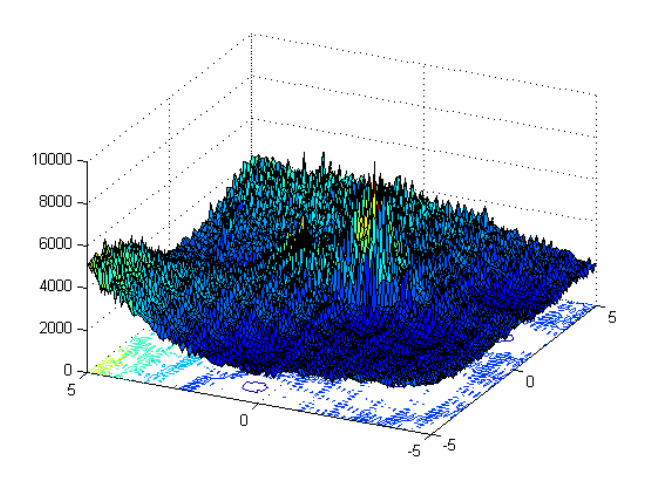

Figure 2-22 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Non-separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Global optimum is on the bound
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{22}(\mathbf{x}^*) = f\_bias_{22}$ =360

Associated Data file:

Name: hybrid_func3_data.mat
hybrid_func3_data.txt
Variable: **o** 10*100 vector the shifted optima for 10 functions
When using, cut **o**=**o**(:,1:*D*)

Name: hybrid_func3_HM_D10 .mat
Variable: **M** an structure variable
Contains **M.M1 M.M2, … , M.M10** ten 10*10 matrixes
Name: hybrid_func3_HM_D10 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 10*10 matrixes, 1-10 lines are **M1**, 11-20 lines are **M2**,....,91-100 lines are **M10**

Name: hybrid_func3_HM_D30 .mat
Variable: **M** an structure variable contains M.**M1,…,M.M10** ten 30*30 matrix
Name: hybrid_func3_MH_D30 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 30*30 matrixes, 1-30 lines are **M1**, 31-60 lines are **M2**,....,271-300 lines are **M10**

Name:	hybrid_func3_MH_D50 .mat
Variable:	**M** an structure variable contains **M.M1,…,M.M10** ten 50*50 matrix
Name:	hybrid_func3_HM_D50 .txt
Variable:	**M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 50*50 matrixes, 1-50 lines are **M1**, 51-100 lines are **M2**,....,451-500 lines are **M10**

2.4.9. F_{23}: *Non-Continuous Rotated Hybrid Composition Function*

All settings are the same as F_{21}.

Except $x_j = \begin{cases} x_j & |x_j - o_{1j}| < 1/2 \\ round(2x_j)/2 & |x_j - o_{1j}| >= 1/2 \end{cases}$ for $j = 1, 2, .., D$

$$round(x) = \begin{cases} a-1 & if \quad x <= 0 \,\&\, b >= 0.5 \\ a & if \quad b < 0.5 \\ a+1 & if \quad x > 0 \,\&\, b >= 0.5 \end{cases},$$

where a is x's integral part and b is x's decimal part

All "round" operators in this document use the same schedule.

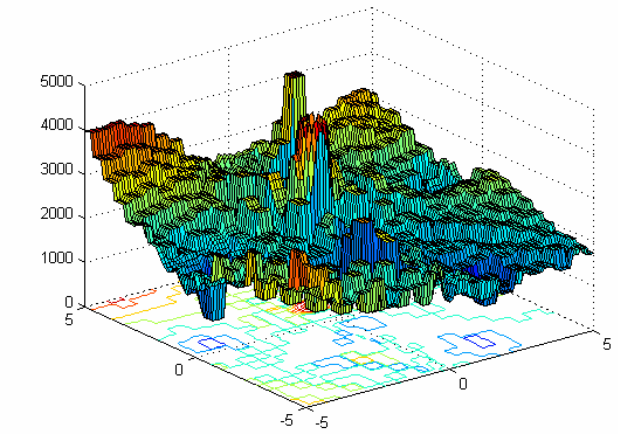

Figure 2-23 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Non-separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Non-continuous
- Global optimum is on the bound
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $f(\boldsymbol{x}^*) \approx$ ***f_bias*** (23)=360

Associated Data file:

Same as F_{21}.

2.4.10. F_{24}: *Rotated Hybrid Composition Function*

$f_1(\mathbf{x})$: Weierstrass Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k (x_i + 0.5))]) - D\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k 0.5)],$$

a=0.5, b=3, k_{max}=20

$f_2(\mathbf{x})$: Rotated Expanded Scaffer's F6 Function

$$F(x,y) = 0.5 + \frac{(\sin^2(\sqrt{x^2+y^2}) - 0.5)}{(1+0.001(x^2+y^2))^2}$$

$$f_i(\mathbf{x}) = F(x_1,x_2) + F(x_2,x_3) + ... + F(x_{D-1},x_D) + F(x_D,x_1)$$

$f_3(\mathbf{x})$: F8F2 Function

$$F8(\mathbf{x}) = \sum_{i=1}^{D}\frac{x_i^{\,2}}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1$$

$$F2(\mathbf{x}) = \sum_{i=1}^{D-1}(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$$

$$f_i(\mathbf{x}) = F8(F2(x_1,x_2)) + F8(F2(x_2,x_3)) + ... + F8(F2(x_{D-1},x_D)) + F8(F2(x_D,x_1))$$

$f_4(\mathbf{x})$: Ackley's Function

$$f_i(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)) + 20 + e$$

$f_5(\mathbf{x})$: Rastrigin's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(x_i^{\,2} - 10\cos(2\pi x_i) + 10)$$

$f_6(\mathbf{x})$: Griewank's Function

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}\frac{x_i^{\,2}}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1$$

$f_7(\mathbf{x})$: Non-Continuous Expanded Scaffer's F6 Function

$$F(x,y) = 0.5 + \frac{(\sin^2(\sqrt{x^2+y^2}) - 0.5)}{(1+0.001(x^2+y^2))^2}$$

$$f(\mathbf{x}) = F(y_1,y_2) + F(y_2,y_3) + ... + F(y_{D-1},y_D) + F(y_D,y_1)$$

$$y_j = \begin{cases} x_j & |x_j| < 1/2 \\ round(2x_j)/2 & |x_j| >= 1/2 \end{cases} \text{ for } j = 1,2,..,D$$

$f_8(\mathbf{x})$: Non-Continuous Rastrigin's Function

$$f(\mathbf{x}) = \sum_{i=1}^{D}(y_i^{\,2} - 10\cos(2\pi y_i) + 10)$$

$$y_j = \begin{cases} x_j & |x_j| < 1/2 \\ round(2x_j)/2 & |x_j| >= 1/2 \end{cases} \text{ for } j = 1,2,..,D$$

$f_9(\mathbf{x})$: High Conditioned Elliptic Function

$$f(\mathbf{x}) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$$

$f_{10}(\mathbf{x})$: Sphere Function with Noise in Fitness

$$f_i(\mathbf{x}) = (\sum_{i=1}^{D} x_i^2)(1 + 0.1|N(0,1)|)$$

$\sigma_i = 2$,for $i = 1,2...,D$

$\boldsymbol{\lambda}$ =[10; 5/20; 1; 5/32; 1; 5/100; 5/50; 1; 5/100; 5/100]

$\mathbf{M}_i$ are all rotation matrices, condition numbers are [100 50 30 10 5 5 4 3 2 2];

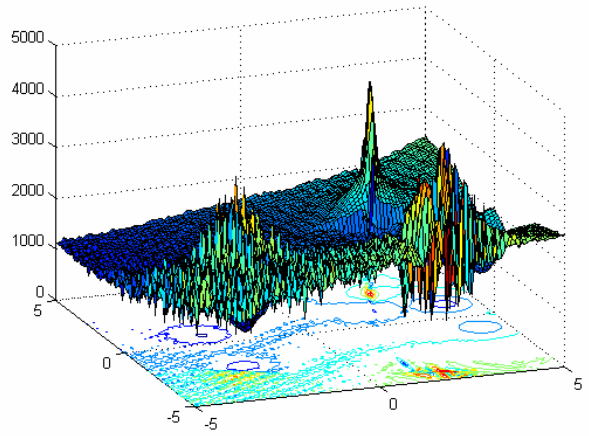

Figure 2-24 3-*D* map for 2-*D* function

Properties:

- Multi-modal
- Rotated
- Non-Separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Unimodal Functions give flat areas for the function.
- $\mathbf{x} \in [-5,5]^D$, Global optimum $\mathbf{x}^* = \mathbf{o}_1$, $F_{24}(\mathbf{x}^*) = f\_bias_{24} = 260$

Associated Data file:

Name: hybrid_func4_data.mat
hybrid_func4_data.txt

Variable: **o** 10*100 vector the shifted optima for 10 functions
When using, cut **o**=**o**(:,1:*D*)

Name: hybrid_func4_M_D10 .mat
Variable: **M** an structure variable
Contains **M.M1 M.M2, … , M.M10** ten 10*10 matrixes
Name: hybrid_func4_M_D10 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 10*10 matrixes, 1-10 lines are **M1**, 11-20 lines are **M2**,....,91-100 lines are **M10**

Name: hybrid_func4_M_D30 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 30*30 matrix
Name: hybrid_func4_M_D30 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 30*30 matrixes, 1-30 lines are **M1**, 31-60 lines are **M2**,....,271-300 lines are **M10**

Name: hybrid_func4_M_D50 .mat
Variable: **M** an structure variable contains **M.M1,…,M.M10** ten 50*50 matrix
Name: hybrid_func4_M_D50 .txt
Variable: **M1 M2 M3 M4 M5 M6 M7 M8 M9 M10** are ten 50*50 matrixes, 1-50 lines are **M1**, 51-100 lines are **M2**,....,451-500 lines are **M10**

2.4.11. F_{25}: *Rotated Hybrid Composition Function without bounds*

All settings are the same as F_{24} except no exact search range set for this test function.

Properties:

- Multi-modal
- Non-separable
- Scalable
- A huge number of local optima
- Different function's properties are mixed together
- Unimodal Functions give flat areas for the function.
- Global optimum is on the bound
- No bounds
- Initialize population in $[2,5]^D$, Global optimum $\boldsymbol{x}^* = \boldsymbol{o}_1$ is outside of the initialization range, $F_{25}(\mathbf{x}^*) = f\_bias_{25}$ =260

Associated Data file:

Same as F_{24}

2.5 Comparisons Pairs

Different Condition Numbers:

- F_1. Shifted Rotated Sphere Function
- F_2. Shifted Schwefel's Problem 1.2
- F_3. Shifted Rotated High Conditioned Elliptic Function

Function With Noise Vs Without Noise

Pair 1:

- F_2. Shifted Schwefel's Problem 1.2
- F_4. Shifted Schwefel's Problem 1.2 with Noise in Fitness

Pair 2:

- F_{16}. Rotated Hybrid Composition Function
- F_{17}. F_{16}. with Noise in Fitness

Function without Rotation Vs With Rotation

Pair 1:

- F_9. Shifted Rastrigin's Function
- F_{10}. Shifted Rotated Rastrigin's Function

Pair 2:

- F_{15}. Hybrid Composition Function
- F_{16}. Rotated Hybrid Composition Function

Continuous Vs Non-continuous

- F_{21}. Rotated Hybrid Composition Function
- F_{23}. Non-Continuous Rotated Hybrid Composition Function

Global Optimum on Bounds Vs Global Optimum on Bounds

- F_{18}. Rotated Hybrid Composition Function
- F_{20}. Rotated Hybrid Composition Function with the Global Optimum on the Bounds

Wide Global Optimum Basin Vs Narrow Global Optimum Basin

- F_{18}. Rotated Hybrid Composition Function
- F_{19}. Rotated Hybrid Composition Function with a Narrow Basin for the Global Optimum

Orthogonal Matrix Vs High Condition Number Matrix

- F_{21}. Rotated Hybrid Composition Function
- F_{22}. Rotated Hybrid Composition Function with High Condition Number Matrix

Global Optimum in the Initialization Range Vs outside of the Initialization Range

- F_{24}. Rotated Hybrid Composition Function
- F_{25}. Rotated Hybrid Composition Function without Bounds

2.6 Similar Groups:

Unimodal Functions

Function 1-5

Multi-modal Functions

Function 6-25

- Single Function: Function 6-12
- Expanded Function: Function 13-14
- Hybrid Composition Function: Function 15-25

Functions with Global Optimum outside of the Initialization Range

- F_7. Shifted Rotated Griewank's Function without Bounds
- F_{25}. Rotated Hybrid Composition Function 4 without Bounds

Functions with Global Optimum on Bounds

- F_5. Schwefel's Problem 2.6 with Global Optimum on Bounds
- F_8. Shifted Rotated Ackley's Function with Global Optimum on Bounds
- F_{20}. Rotated Hybrid Composition Function 2 with the Global Optimum on the Bounds

3. Evaluation Criteria

3.1 Description of the Evaluation Criteria

Problems: 25 minimization problems

Dimensions: D=10, 30, 50

Runs / problem: 25 (**Do not run many 25 runs to pick the best run**)

Max_FES: 10000*D (Max_FES_10D= 100000; for 30D=300000; for 50D=500000)

Initialization: Uniform random initialization within the search space, except for problems 7 and 25, for which initialization ranges are specified.

Please use the same initializations for the comparison pairs (problems 1, 2, 3 & 4, problems 9 & 10, problems 15, 16 & 17, problems 18, 19 & 20, problems 21, 22 & 23, problems 24 & 25). One way to achieve this would be to use a fixed seed for the random number generator.

Global Optimum: All problems, except 7 and 25, have the global optimum within the given bounds and there is no need to perform search outside of the given bounds for these problems. 7 & 25 are exceptions without a search range and with the global optimum outside of the specified initialization range.

Termination: Terminate before reaching Max_FES if the error in the function value is 10^{-8} or less.

Ter_Err: 10^{-8} (termination error value)

1) **Record function error value (f(x)-f(x*)) after 1e3, 1e4, 1e5 FES and at termination (due to Ter_Err or Max_FES) for each run.**

   For each function, sort the error values in 25 runs from the smallest (best) to the largest (worst**)**

   Present the following: 1^{st} (best), 7^{th}, 13^{th} (median), 19^{th}, 25^{th} (worst) function values

   Mean and STD for the 25 runs

2) **Record the FES needed in each run to achieve the following fixed accuracy level. The Max_FES applies.**

Table 3-1 Fixed Accuracy Level for Each Function

Function	Accuracy	Function	Accuracy
1	-450 + 1e-6	14	-300 + 1e-2

2	-450 + 1e-6	15	120 + 1e-2
3	-450 + 1e-6	16	120 + 1e-2
4	-450 + 1e-6	17	120 + 1e-1
5	-310 + 1e-6	18	10+ 1e-1
6	390 + 1e-2	19	10 + 1e-1
7	-180 + 1e-2	20	10 + 1e-1
8	-140 + 1e-2	21	360 + 1e-1
9	-330 + 1e-2	22	360 + 1e-1
10	-330 + 1e-2	23	360 + 1e-1
11	90 + 1e-2	24	260 + 1e-1
12	-460 + 1e-2	25	260 + 1e-1
13	-130 + 1e-2		

Successful Run: A run during which the algorithm achieves the fixed accuracy level within the Max_FES for the particular dimension.

For each function/dimension, sort FES in 25 runs from the smallest (best) to the largest (worst**)**

Present the following: 1^{st} (best), 7^{th}, 13^{th} (median), 19^{th}, 25^{th} (worst) FES

Mean andSTD for the 25 runs

3) Success Rate & success Performance For Each Problem

Success Rate**= (#** of successful runs according to the table above) / total runs

Success Performance=mean (FEs for successful runs)*(# of total runs) / (# of successful runs)

The above two quantities are computed for each problem separately.

4) Convergence Graphs (or Run-length distribution graphs)

Convergence Graphs for each problem for ***D*=30**. The graph would show the median performance of the total runs with termination by either the Max_FES or the Ter_Err. The semi-log graphs should show log10(f(***x***)- f(***x****)) vs FES for each problem.

5) Algorithm Complexity

a) Run the test program below:

for i=1:1000000

x= (double) 5.55;

x=x + x; *x=x./*2; *x=x*x; x=sqrt(x); x=ln(x); x=exp(x); y=x/x;*

end

Computing time for the above=*T0*;

b) evaluate the computing time just for Function 3. For 200000 evaluations of a certain dimension *D*, it gives T1;

c) the complete computing time for the algorithm with 200000 evaluations of the same *D* dimensional benchmark function 3 is *T2*. Execute step c 5 times and get 5 *T2* values. $\hat{T}2$ =Mean(*T2*)

The complexity of the algorithm is reflected by: $\hat{T}2$, *T1*, *T0*, and ($\hat{T}2$ *-T1)/T0*

The algorithm complexities are calculated on 10, 30 and 50 dimensions, to show the algorithm complexity's relationship with dimension. Also provide sufficient details on the computing system and the programming language used. In step c, we execute the complete algorithm 5 times to accommodate variations in execution time due adaptive nature of some algorithms.

6) Parameters

We discourage participants searching for a distinct set of parameters for each problem/dimension/etc. Please provide details on the following whenever applicable:

a) All parameters to be adjusted

b) Corresponding dynamic ranges

c) Guidelines on how to adjust the parameters

d) Estimated cost of parameter tuning in terms of number of FEs

e) Actual parameter values used.

7) Encoding

If the algorithm requires encoding, then the encoding scheme should be independent of the specific problems and governed by generic factors such as the search ranges.

3.2 Example

System: Windows XP (SP1)

CPU: Pentium(R) 4 3.00GHz

RAM: 1 G

Language: Matlab 6.5

Algorithm: Particle Swarm Optimizer (PSO)

Results

***D*=10**
Max_FES=100000

Table 3-2 Error Values Achieved When FES=1e3, FES=1e4, FES=1e5 for Problems 1-8

FES \ Prob		1	2	3	4	5	6	7	8
1e3	1st(Best)	4.8672e+2	4.7296e+2	2.2037e+6	4.6617e+2	2.3522e+3			
	7th	8.0293e+2	9.8091e+2	8.5141e+6	1.2900e+3	4.0573e+3			
	13th(Median)	9.2384e+2	1.5293e+3	1.4311e+7	1.9769e+3	4.6308e+3			
	19th	1.3393e+3	1.7615e+3	1.9298e+7	2.9175e+3	4.8015e+3			
	25th (Worst)	1.9151e+3	3.2337e+3	4.4688e+7	6.5038e+3	5.6701e+3			
	Mean	1.0996e+3	1.5107e+3	1.5156e+7	2.3669e+3	4.4857e+3			
	Std	4.0575e+2	7.2503e+2	9.3002e+6	1.5082e+3	7.0081e+2			
1e4	1st(Best)	3.1984e-3	1.0413e+0	1.3491e+5	6.7175e+0	1.6584e+3			
	7th	2.6509e-2	1.3202e+1	4.4023e+5	3.8884e+1	2.3522e+3			
	13th(Median)	6.0665e-2	1.9981e+1	1.1727e+6	5.5027e+1	2.6335e+3			
	19th	1.0657e-1	3.5319e+1	2.0824e+6	7.1385e+1	2.8788e+3			
	25th (Worst)	4.3846e-1	1.0517e+2	2.9099e+6	1.7905e+2	3.6094e+3			
	Mean	8.6962e-2	2.7883e+1	1.3599e+6	5.9894e+1	2.6055e+3			
	Std	9.6616e-2	2.3526e+1	9.1421e+5	3.5988e+1	4.5167e+2			
1e5	1st(Best)	4.7434e-9T	5.1782e-9T	4.2175e+4	1.7070e-5	1.1864e+3			
	7th	7.9845e-9T	8.5278e-9T	1.2805e+5	1.2433e-3	1.4951e+3			
	13th(Median)	9.0901e-9T	9.7281e-9T	2.3534e+5	4.0361e-3	1.7380e+3			
	19th	9.6540e-9T	1.5249e-8	4.6436e+5	1.8283e-2	1.9846e+3			
	25th (Worst)	9.9506e-9T	2.3845e-7	2.2776e+6	3.9795e-1	2.3239e+3			
	Mean	8.5375e-9T	3.2227e-8	4.6185e+5	3.4388e-2	1.7517e+3			
	Std	1.4177e-9T	6.2340e-8	5.4685e+5	8.2733e-2	2.9707e+2			

*** xxx.e-9T means it get termination error before it gets the predefined record FES.**

Table 3-3 Error Values Achieved When FES=1e+3, FES=1e+4, FES=1e+5 for Problems 9-17

FES \ Prob		9	10	11	12	13	14	15	16	17
1e+3	1st(Best)									
	7th									
	13th(Median)									
	19th									
	25th (Worst)									
	Mean									
	Std									
1e+4	1st(Best)									
	7th									
	13th(Median)									
	19th									
	25th (Worst)									
	Mean									
	Std									
1e+5	1st(Best)									
	7th									
	13th(Median)									
	19th									
	25th (Worst)									
	Mean									
	Std									

Table 3-4 Error Values Achieved When FES=1e+3, FES=1e+4, FES=1e+5 for Problems 18-25

FES \ Prob		18	19	20	21	22	23	24	25
1e+3	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								
1e+4	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								
1e+5	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								

Table 3-5 Number of FES to achieve the fixed accuracy level

Prob	1st(Best)	7th	13th (Median)	19th	25th (Worst)	Mean	Std	Success rate	Success Performance
1	11607	12133	12372	12704	13022	1.2373e+4	3.6607e+2	100%	1.2373e+4
2	17042	17608	18039	18753	19671	1.8163e+4	7.5123e+2	100%	1.8163e+4
3	-	-	-	-	-	-	-	0%	-
4	-	-	-	-	-	-	-	0%	-
5	-	-	-	-	-	-	-	0%	-
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									

D=30
Max_FES=300000

Table 3-6 Error Values Achieved When FES=1e3, FES=1e4, FES=1e5 for Problems 1-8

FES \ Prob		1	2	3	4	5	6	7	8
1e3	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								
1e4	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								
1e5	1st(Best)								
	7th								
	13th(Median)								
	19th								

	25th (Worst)								
	Mean								
	Std								
3e5	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								

……
……

***D*=50**
Max_FES=500000

Table 3-7 Error Values Achieved When FES=1e3, FES=1e4, FES=1e5 for Problems 1-8

FES \ Prob		1	2	3	4	5	6	7	8
1e3	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								
1e4	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								
1e5	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								
3e5	1st(Best)								
	7th								
	13th(Median)								
	19th								
	25th (Worst)								
	Mean								
	Std								

……
……

Convergence Graphs (30D)

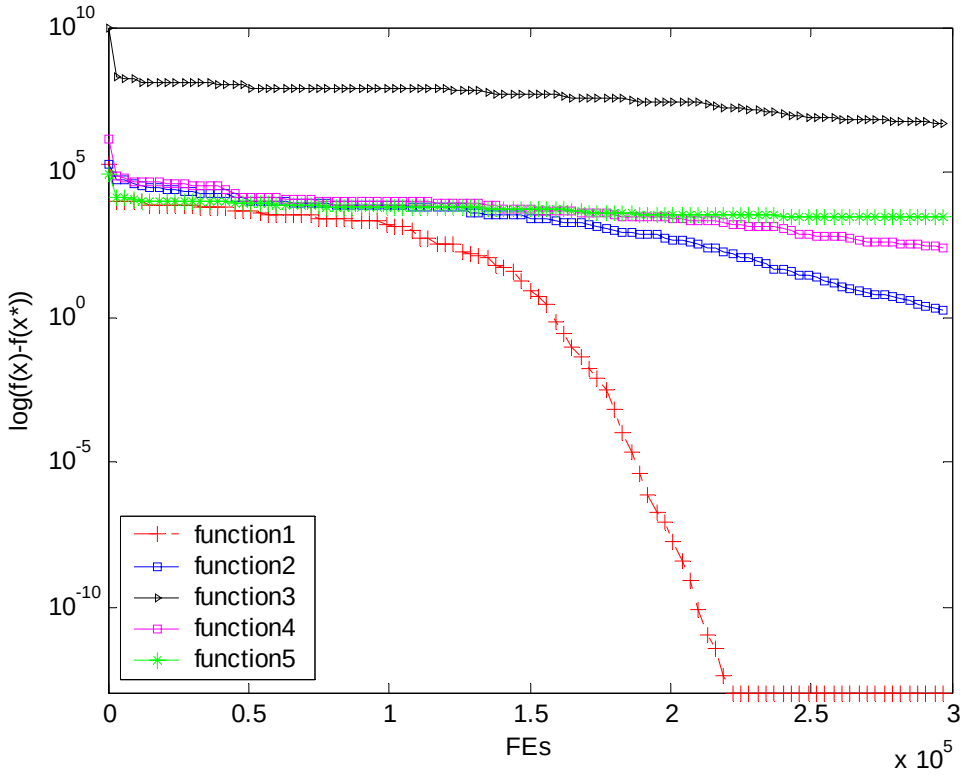

Figure 3-1 Convergence Graph for Functions 1-5

…

Figure 3-2 Convergence Graph for Function 6-10

…

Figure 3-3 Convergence Graph for Function 11-14

…

Figure 3-4 Convergence Graph for Function 15-20

…

Figure 3-5 Convergence Graph for Function 21-25

Algorithm Complexity

Table 3-8 Computational Complexity

	T0	*T1*	$\hat{T}2$	$(\hat{T}2-T1)/T0$
D=10	39.5470	31.1250	82.3906	1.2963
D=30		38.1250	90.8437	1.3331
D=50		46.0780	108.9094	1.5888

Parameters

a) All parameters to be adjusted

b) Corresponding dynamic ranges

c) Guidelines on how to adjust the parameters

d) Estimated cost of parameter tuning in terms of number of FES

e) Actual parameter values used.

4. Notes

Note 1: Linear Transformation Matrix

$$\mathbf{M}=\mathbf{P}*\mathbf{N}*\mathbf{Q}$$

P, **Q** are two orthogonal matrixes, generated using Classical Gram-Schmidt method
N is diagonal matrix

$$u = rand(1, D), d_{ii} = c^{\frac{u_i - \min(u)}{\max(u) - \min(u)}}$$

M's condition number Cond(**M**)=c

Note 2: On page 17, w_i values are sorted and raised to a higher power. The objective is to ensure that each optimum (local or global) is determined by only one function while allowing a higher degree of mixing of different functions just a very short distance away from each optimum.

Note 3: We assign different positive and negative objective function values, instead of zeros. This may influence some algorithms that make use of the objective values.

Note 4: We assign the same objective values to the comparison pairs in order to make the comparison easier.

Note 5: High condition number rotation may convert a multimodal problem into a unimodal problem. Hence, moderate condition numbers were used for multimodal.

Note 6: Additional data files are provided with some coordinate positions and the corresponding fitness values in order to help the verification process during the code translation.

Note 7: It is insufficient to make any statistically meaningful conclusions on the pairs of problems as each case has at most 2 pairs. We would probably require 5 or 10 or more pairs for each case. We would consider this extension for the edited volume.

Note 8: Pseudo-real world problems are available from the web link given below. If you have any queries on these problems, please contact Professor Darrell Whitley directly. Email: whitley@CS.ColoState.EDU
Web-link: http://www.cs.colostate.edu/~genitor/functions.html.

Note 9: We are recording the numbers such as 'the number of FES to reach the given fixed accuracy', 'the objective function value at different number of FES' **for each run of each problem and each dimension** in order to perform some statistical significance tests. The details of a statistical significance test would be made available a little later.

References:

[1] N. Hansen, S. D. Muller and P. Koumoutsakos, "Reducing the Time Complexity of the Derandomized evolution Strategy with Covariance Matrix Adaptation (CMA-ES)." *Evolutionary Computation*, 11(1), pp. 1-18, 2003

[2] A. Klimke, "Weierstrass function's matlab code", http://matlabdb.mathematik.uni-stuttgart.de/download.jsp?MC_ID=9&MP_ID=56

[3] H-P. Schwefel, "Evolution and Optimum Seeking", http://ls11-www.cs.uni-dortmund.de/lehre/wiley/

[4] D. Whitley, K. Mathias, S. Rana and J. Dzubera, "Evaluating Evolutionary Algorithms" *Artificial Intelligence*, 85 (1-2): 245-276 AUG 1996.

Apéndice E

Ejemplo batería de funciones

En este apéndice se quiere mostrar un ejemplo de selección de varias funciones entre el grupo de 25 funciones del PDEC-05, para el caso de que nos desee probar todas las funciones, siempre bajo la premisa de que el grupo sea lo más completo y homogéneo posible.

E.1. Selección

Los criterios seguidos para la selección de entre todas las funciones del PDEC-05 son los siguientes:

- Popularidad (fuentes en las que se menciona la función, Nº de resultados en Google Académico[1]...)
- Implementación de esta en software
- Conjunto que abarque todas las características de funciones (ver apartado D.1.0.5)
- Técnicas para cambiar el escenario variables/dimensión (apartado D.2)

En total son diez funciones de prueba cuya función objetivo es de minimización. Se ha elegido este número con el fin de conseguir un conjunto no demasiado extenso que no cumpliría con el requisito de reducir la batería, ni muy reducido que podría no englobar todas la características mencionadas en la sección anterior.

[1] http://scholar.google.es/schhp?hl=es

Las funciones escogidas son:

- f_1: Función Esfera desplazada (De Jong 1 desplazada).
- f_2: Función de Rosenbrock desplazada (De Jong 2 desplazada).
- f_3: Función extendida de Scaffer 6 desplazada y rotada.
- f_4: Función Elíptica desplazada, rotada y altamente condicionada.
- f_5: Función de Rastrigin desplazada.
- f_6: Función de Ackley rotada y desplazada con óptimo global en la frontera.
- f_7: Función híbrida compuesta por:
  - Función de Rastrigin.
  - Función de Weierstass.
  - Función de Griewank.
  - Función de Ackley.
  - Función Esfera.
- f_8: Función f_7 (híbrida) rotada.
- f_9: Función f_8 (híbrida) con ruido.
- f_{10}: Función híbrida no continua compuesta por:
  - Función rotada y extendida de Scaffer 6 f_3.
  - Función de Rastrigin f_5.
  - Función de Ackley f_6 con función de Rosenbrock f_2 (función expandida).
  - Función de Weierstrass.
  - Función de Griewank.

Del conjunto de las 25 funciones del PDEC-05, se han escogido, debido a su popularidad, las 2 funciones de De Jong disponibles. Las 5 funciones de De Jong figuran en un amplio número de documentos y estudios sobre computación evolutiva. Además ya fueron incluidas por Goldberg [Gol89a] en su libro, donde se describen sus características y se muestran algunos resultados a modo de ejemplo. Estos son algunos de los motivos por lo que resulte casi de obligada

necesidad incluirlas en cualquier batería de pruebas.

La función de Scaffer 6, Rastrigin y Ackley también gozan de buena popularidad y lo demuestran apareciendo un gran número de resultados en los buscadores más famosos. La función Elíptica se ha incluido ya que es necesaria para calcular la complejidad del algoritmo en el procedimiento de evaluación.

Tampoco se ha perdido de vista elegir funciones de cada conjunto que presenta el PDEC-05 (funciones Unimodales, funcions multimodáles básicas, funciones multimodales extendidas y funciones compuestas) para así mantener un grupo homogéneo. Por lo tanto se explica así la inclusión de las 3 funciones híbridas (f_7, f_8 y f_9), ya que recogen las características de todas las funciones que las componen.

Por último, con el fin de disponer de una función discontinua (una de las condiciones reflejadas en el apartado D.1.0.5) se ha escogido la función f_{10}, que es una función híbrida no continua compuesta por la mayoría de las funciones del grupo, lo que permitirá ver mejor cómo reacciona el algoritmo frente a discontinuidades.

En lo referente a las técnicas para cambiar el escenario (rotaciones, desplazamiento, ruido...) se ha buscado también que en el grupo de las 10 funciones escogidas, se recojan todas las técnicas disponibles (explicadas en el apartado D.2).

Por lo tanto, en lo que se refiere a las características que reúne el conjunto de 10 funciones, tenemos:

- Funciones de minimización (en todos los casos se busca minimizar la función objetivo).
- De baja/alta dimensionalidad que se obtiene por medio de la definición del número de variables en el procedimiento de evaluación que va desde 10 pasando por 30 hasta 50 de dimensión.
- Funciones continuas / discontinuas / convexas / no convexas (función Esfera, Rosenbrock, Scaffer 6, ...).
- Funciones unimodales / multimodales.
- Desplazadas y rotadas.
- Con óptimo en la frontera (función de Ackley).
- Funciones con ruido (función híbrida f_9).

E.2. Descripción de las funciones

A continuación se representan y definen las 10 funciones así como sus características:

f_1: Función Esfera desplazada (De Jong 1 desplazada)

$$F_1(\mathbf{x}) = \sum_{i=1}^{D} {z_i}^2 + f\_bias_1; \tag{E.1}$$

donde: $\mathbf{z} = \mathbf{x} - \mathbf{o}$, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$
D : *dimensiones* $\mathbf{o} = [o_1, o_2, \ldots, o_D]$: *el óptimo global desplazado*

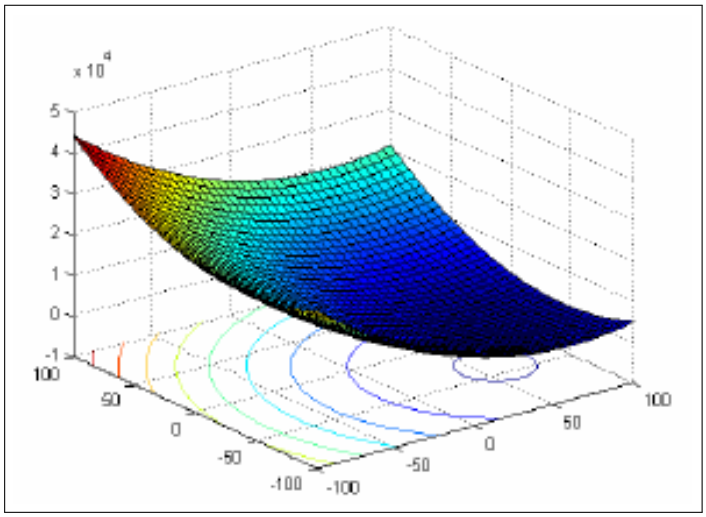

Figura E.1: Representación invertida 3D de la función F1 en 2D [SHL+05]

Propiedades:

- Unimodal.
- Desplazada.
- Separable.
- Escalable.
- $\mathbf{x} \in [-100, 100]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_1(\mathbf{x}^*) = f\_bias_1 = -450$

f_2: Función de Rosenbrock desplazada (De Jong 2 desplazada)

$$F_2(\mathbf{x}) = \sum_{i=1}^{D-1} (100 \cdot (z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f\_bias_6; \qquad \text{(E.2)}$$

donde: $\mathbf{z} = \mathbf{x} - \mathbf{o} + 1, \quad \mathbf{x} = [x_1, x_2, \ldots, x_D]$

$D : dimensiones \quad \mathbf{o} = [o_1, o_2, \ldots, o_D] : el\ óptimo\ global\ desplazado$

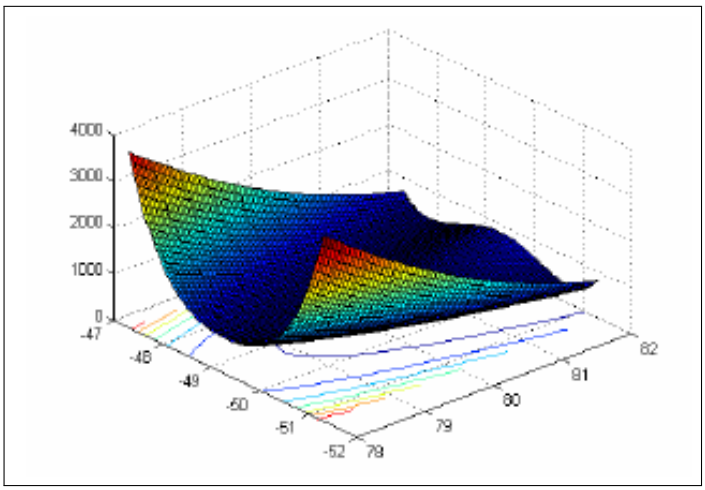

Figura E.2: Representación invertida 3D de la función F2 en 2D [SHL+05]

Propiedades:

- Multimodal.
- Desplazada.
- No separable.
- Escalable.
- Presenta un estrecho valle desde el óptimo local al global.
- $\mathbf{x} \in [-100, 100]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_2(\mathbf{x}^*) = f\_bias_6 = 390$

f_3: Función extendida de Scaffer 6 desplazada y rotada

$$F(x,y) = 0,5 + \frac{\sin^2(\sqrt{x^2+y^2}) - 0,5}{(1+0,001\cdot(x^2+y^2))^2}; \tag{E.3}$$

$$F_3(\mathbf{x}) = EF(z_1, z_2, \ldots, z_D) = F(z_1, z_2) + F(z_2, z_3) + \ldots + F(z_{D-1}, z_D) + F(z_D, z_1) + f\_bias_{14}; \tag{E.4}$$

donde: $\mathbf{z} = (\mathbf{x} - \mathbf{o}) \cdot M, \quad \mathbf{x} = [x_1, x_2, \ldots, x_D]$
D : *dimensiones* $\quad \mathbf{o} = [o_1, o_2, \ldots, o_D]$: *el óptimo global desplazado*
$\mathbf{M}$: *matriz de transformación lineal*, $\quad$ *número de condición* $= 3$

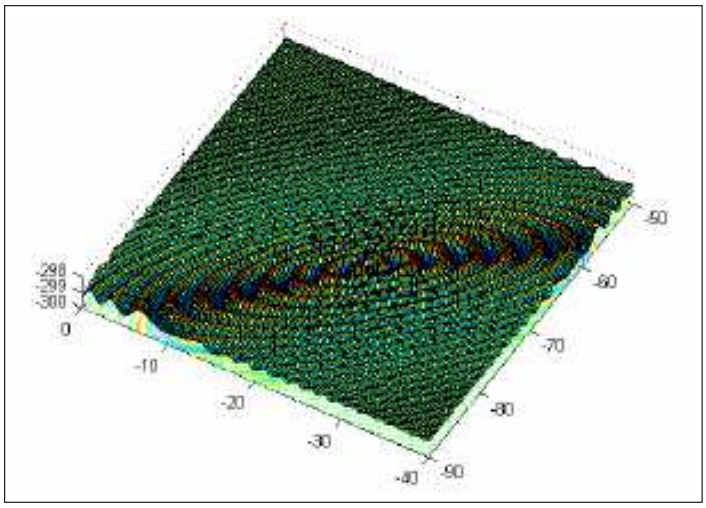

Figura E.3: Representación invertida 3D de la función F3 en 2D [SHL$^+$05]

Propiedades:

- Multimodal.
- Desplazada.
- No separable. .
- Escalable.
- $\mathbf{x} \in [-100, 100]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_3(\mathbf{x}^*) = f\_bias_{14} = -300$

f_4: Función Elíptica desplazada, rotada y altamente condicionada

$$F_4(\mathbf{x}) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} {z_i}^2 + f\_bias_3; \tag{E.5}$$

donde: $\mathbf{z} = \mathbf{x} - \mathbf{o} \cdot \mathbf{M}, \quad \mathbf{x} = [x_1, x_2, \ldots, x_D]$
D : $dimensiones$ $\quad \mathbf{o} = [o_1, o_2, \ldots, o_D]$: $el\ óptimo\ global\ desplazado$
$\mathbf{M}$: $matriz\ ortogonal$

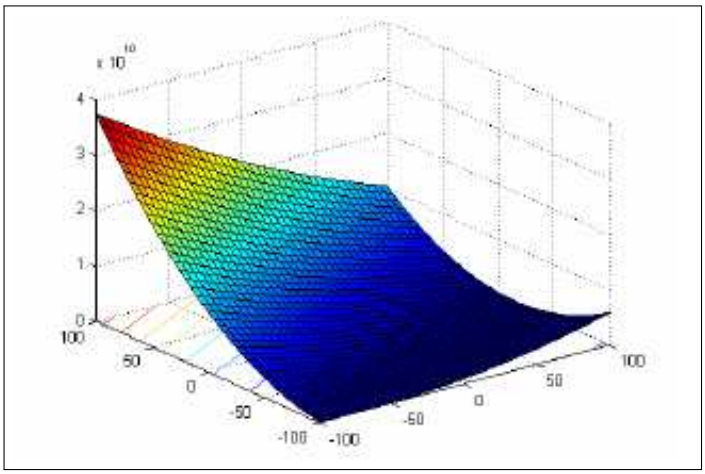

Figura E.4: Representación invertida 3D de la función F4 en 2D [SHL$^+$05]

Propiedades:

- Unimodal.
- Desplazada.
- Rotada.
- No separable.
- Escalable.
- $\mathbf{x} \in [-100, 100]^D$, Óptimo Global: $\mathbf{x}^* = \mathbf{o}$, $F_4(\mathbf{x}^*) = f\_bias_3 = -450$

f_5: Función de Rastrigin desplazada

$$F_5(\mathbf{x}) = \sum_{i=1}^{D}({z_i}^2 - 10 \cdot \cos(2\pi z_i) + 10) + f\_bias_9; \tag{E.6}$$

donde: $\mathbf{z} = \mathbf{x} - \mathbf{o}$, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$
$D : dimensiones$ $\mathbf{o} = [o_1, o_2, \ldots, o_D] : el\ óptimo\ global\ desplazado$

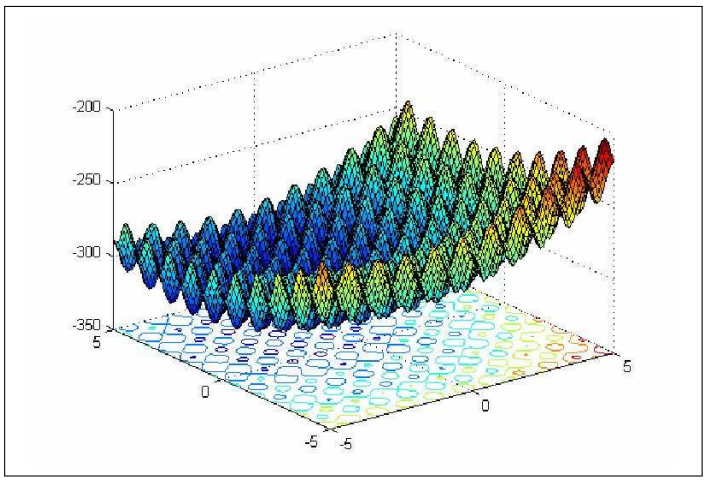

Figura E.5: Representación invertida 3D de la función F5 en 2D [SHL$^+$05]

Propiedades:

- Multimodal.
- Desplazada.
- Separable.
- Escalable.
- Gran número de óptimos locales.
- $\mathbf{x} \in [-5, 5]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_5(\mathbf{x}^*) = f\_bias_9 = -330$

f_6: Función de Ackley rotada y desplazada con óptimo global en la frontera

$$F_6(\mathbf{x}) = -20 \cdot exp\left(-0{,}2 \cdot \sqrt{\frac{1}{D}\sum_{i=1}^{D} {z_i}^2}\right) - exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right) + 20 + e + f\_bias_8; \quad \text{(E.7)}$$

donde: $\mathbf{z} = \mathbf{x} - \mathbf{o} \cdot \mathbf{M}, \quad \mathbf{x} = [x_1, x_2, \ldots, x_D]$

$D : dimensiones \quad \mathbf{o} = [o_1, o_2, \ldots, o_D] : el\ óptimo\ global\ desplazado$

Después de cargar el fichero de datos, el ajuste $o_{2j-1} = -32 \cdot o_{2j}$ es aleatoriamente ubicado en el espacio de búsqueda (en $j = 1, 2, \ldots, \lfloor D/2 \rfloor$).

$\mathbf{M} : \ matriz\ de\ transformación\ lineal, \quad número\ de\ condición = 100$

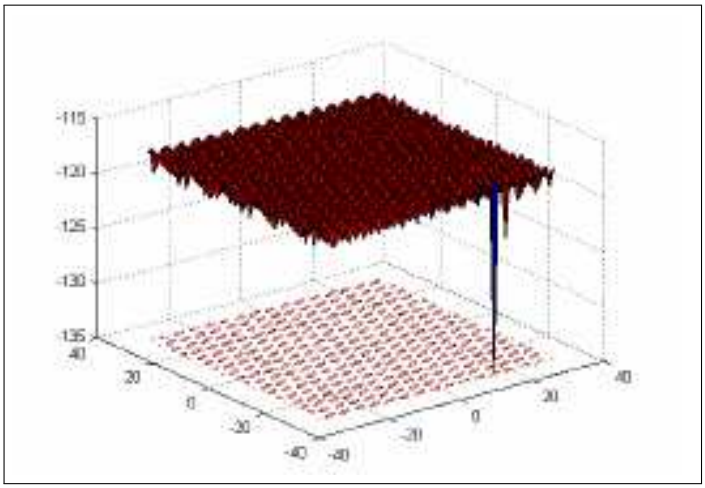

Figura E.6: Representación invertida 3D de la función F6 en 2D [SHL+05]

Propiedades:

- Multimodal.
- Rotada.
- Desplazada.
- No separable.
- Escalable.

- El número de condición de **A** aumenta con el número de variables $O(D^2)$.
- Óptimo global en la frontera.
- Si en el proceso de inicialización hay población en la frontera, el problema alcanzará el óptimo de un modo fácil.
- $\mathbf{x} \in [-32, 32]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_6(\mathbf{x}^*) = f\_bias_8 = -140$

f_7: Función híbrida Para consultar la información de cómo se construyen las funciones híbridas compuestas, por favor acuda al apartado D.3.3.

F_7 compuesta por[2]:

- $f_{1-2}(\mathbf{x})$: Función de Rastrigin

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}({x_i}^2 - 10 \cdot \cos(2\pi x_i) + 10); \tag{E.8}$$

- $f_{3-4}(\mathbf{x})$: Función de Weierstrass

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(\sum_{k=0}^{k_{max}}[a^k \cdot \cos(2\pi b^k \cdot (x_i + 0,5))]) - D\sum_{k=0}^{k_{max}}[a^k \cdot \cos(2\pi b^k \cdot 0,5)]; \tag{E.9}$$

$$\text{donde:} \qquad a = 0,5 \quad b = 3, \quad k_{max} = 20$$

- $f_{5-6}(\mathbf{x})$: Función de Griewank

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1; \tag{E.10}$$

- $f_{7-8}(\mathbf{x})$: Función de Ackley

$$f_i(\mathbf{x}) = -20 \cdot exp\left(-0{,}2 \cdot \sqrt{\frac{1}{D}\sum_{i=1}^{D}{x_i}^2}\right) - exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e; \tag{E.11}$$

- $f_{9-10}(\mathbf{x})$: Función Esfera

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}{x_i}^2; \tag{E.12}$$

[2]Las formulas aquí representadas son funciones básicas. Por lo tanto rotación y traslación no están incluidas en estas expresiones.

$\sigma_i = 1 \quad para \quad i = 1, 2, \ldots, D$
$\lambda = [1, 1, 10, 10, 5/60, 5/60, 5/32, 5/32, 5/100, 5/100]$
M_i: Matrices identidad

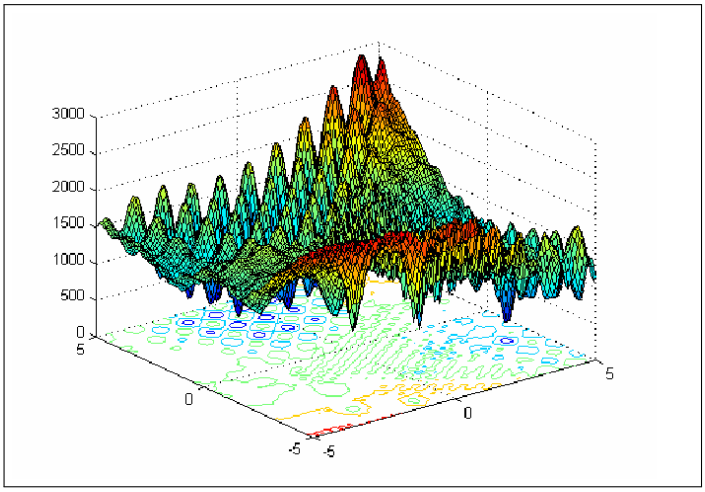

Figura E.7: Representación invertida 3D de la función F7 en 2D [SHL+05]

Propiedades:

- Multimodal.
- Separable próxima al óptimo global (Rastrigin).
- Escalable.
- Gran número de óptimos locales.
- Mezcla y combina diferentes propiedades de diferentes funciones.
- La función Esfera proporciona dos áreas planas en la función.
- $\mathbf{x} \in [-5, 5]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_7(\mathbf{x}^*) = f\_bias_{15} = 120$

f_8: Función f_7 (híbrida) rotada Idéntica a F_7, salvo que las matrices de transformación lineal $\mathbf{M}_i$ son diferentes.

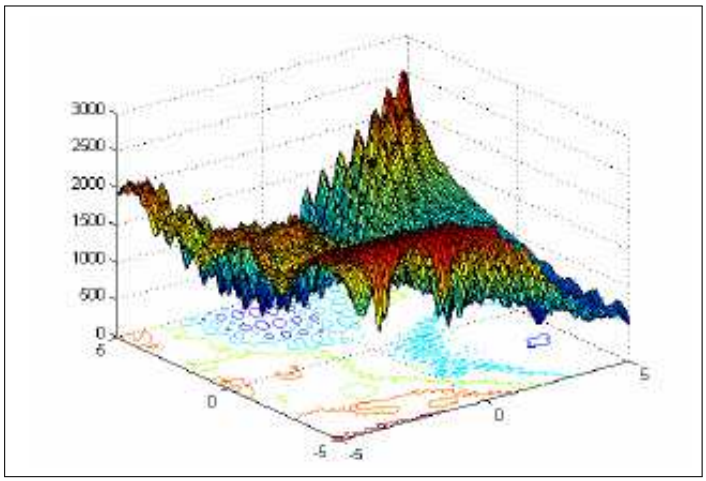

Figura E.8: Representación invertida 3D de la función F8 en 2D [SHL+05]

Propiedades:

- Multimodal.
- Rotada.
- No separable.
- Escalable.
- Gran número de óptimos locales.
- Mezcla y combina diferentes propiedades de diferentes funciones.
- La función Esfera proporciona dos áreas planas en la función.
- $\mathbf{x} \in [-5, 5]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_8(\mathbf{x}^*) = f\_bias_{16} = 120$

f_9: Función f_8 (híbrida) con ruido

$$G(\mathbf{x}) = F_8 - f\_bias_16; \tag{E.13}$$

$$F_9(\mathbf{x}) = G(\mathbf{x}) \cdot (1 + 0,2|N(0,1)|) + f\_bias_{17}; \tag{E.14}$$

Todo el resto de parámetros son iguales a F_8.

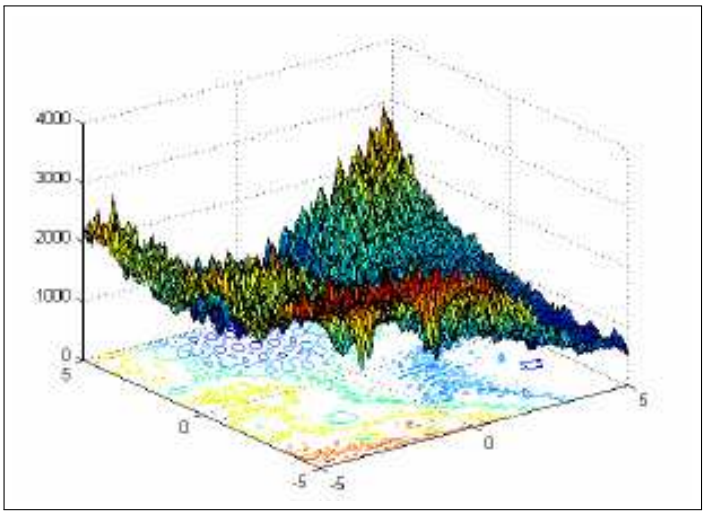

Figura E.9: Representación invertida 3D de la función F9 en 2D [SHL$^+$05]

Propiedades:

- Multimodal.
- Rotada.
- No separable.
- Escalable.
- Gran número de óptimos locales.
- Mezcla y combina diferentes propiedades de diferentes funciones.
- La función Esfera proporciona dos áreas planas en la función.
- Ruido gausiano.
- $\mathbf{x} \in [-5, 5]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_9(\mathbf{x}^*) = f\_bias_{17} = 120$

f_{10}: Función híbrida no continua F_{10} compuesta por[3]:

- $f_{1-2}(\mathbf{x})$:Función y extendida de Scaffer 6

$$F(x,y) = 0,5 + \frac{\sin^2(\sqrt{x^2+y^2}) - 0,5}{(1+0,001\cdot(x^2+y^2))^2}; \tag{E.15}$$

$$f_i(\mathbf{x}) = F(x_1,x_2) + F(x_2,x_3) + \cdots + F(x_{D-1},x_D) + F(x_D,x_1); \tag{E.16}$$

- $f_{3-4}(\mathbf{x})$: Función de Rastrigin

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}({x_i}^2 - 10\cdot\cos(2\pi x_i) + 10); \tag{E.17}$$

- $f_{5-6}(\mathbf{x})$: Función de Ackley f_6 con función de Rosenbrock f_2 (función expandida[4])

$$f_6(\mathbf{x}) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1; \tag{E.18}$$

$$f_2(\mathbf{x}) = \sum_{i=1}^{D-1}(100\cdot(x_i^2 - x_{i+1})^2 + (x_i-1)^2); \tag{E.19}$$

$$\begin{aligned} f_i(\mathbf{x}) = {} & f_6(f_2(x_1,x_2)) + f_6(f_2(x_2,x_3)) + \ldots \\ & + f_6(f_2(x_{D-1},x_D)) + f_6(f_2(x_D,x_1)); \end{aligned} \tag{E.20}$$

- $f_{7-8}(\mathbf{x})$: Función de Weierstrass

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}(\sum_{k=0}^{k_{max}}[a^k\cdot\cos(2\pi b^k\cdot(x_i+0,5))]) - D\sum_{k=0}^{k_{max}}[a^k\cdot\cos(2\pi b^k\cdot 0,5)]; \tag{E.21}$$

- $f_{9-10}(\mathbf{x})$: Función de Griewank

$$f_i(\mathbf{x}) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1; \tag{E.22}$$

[3]Las formulas aquí representadas son funciones básicas. Por lo tanto rotación y traslación no están incluidas en estas expresiones.

[4]Para consultar la información de cómo se construyen las funciones expandidas, por favor acuda al apartado D.3.3.

$\sigma = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]$
$\lambda = [5 * 5/100;\ 5/100;\ 5 * 1;\ 1;\ 5 * 10;\ 10;\ 5 * 5/200;\ 5/200]$
M_i: Matrices ortogonales

La función híbrida f_{10} se compone del mismo modo que la función híbrida f_7 (la información de cómo se construyen estas se encuentra en el apartado D.3.3), excepto:

$$x_j = \begin{cases} x_j & | \ x_j - o_{1j} \ | < 1/2 \\ round(2x_j)/2 & | \ x_j - o_{1j} \ | >= 1/2 \end{cases} \quad \text{for } j = 1, 2, \ldots, D \tag{E.23}$$

$$round(x) = \begin{cases} a - 1 & \text{if} \quad x <= 0 \ \& \ b >= 0,5 \\ a & \text{if} \quad b < 0,5 \\ a & \text{if} \quad x > 0 \ \& \ b >= 0,5 \end{cases} \tag{E.24}$$

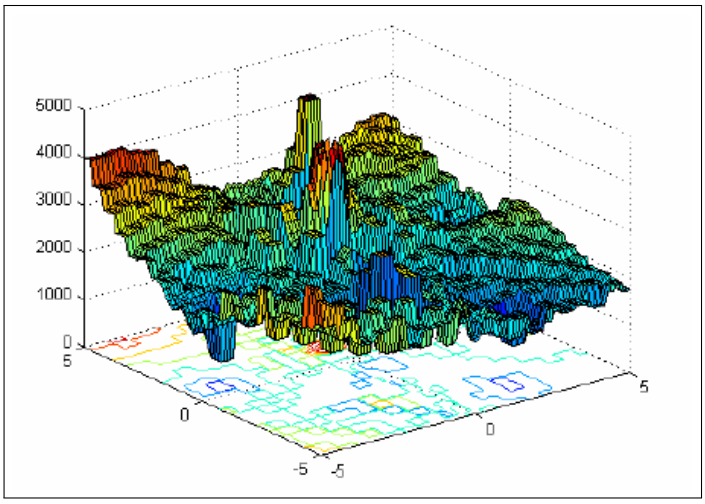

Figura E.10: Representación invertida 3D de la función F10 en 2D [SHL+05]

Propiedades:

- Multimodal.
- No separable.
- Escalable.

- Gran número de óptimos locales.
- Mezcla y combina diferentes propiedades de diferentes funciones.
- Discontinua.
- Óptimo global en la frontera.
- $\mathbf{x} \in [-5, 5]^D$, Óptimo Global: $x^* = \mathbf{o}$, $F_10(\mathbf{x}^*) \approx f\_bias_{23} = 360$

Apéndice F

Resultados obtenidos

A continuación se muestra una relación, bajo los criterios del PDEC-05 (apéndice D), de los resultados obtenidos en la experimentación, capítulo 5. Además se han recogido algunas gráficas adicionales como complemento a estos criterios.

F.1. Resultados

F.1.1. Grupo de escenarios 10.D.**.0005

Se trata de los estudios sobre el grupo de escenarios 10.D.**.0005, con el fin de determinar el tamaño de población.

1. Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (Function Evaluations) en la finalización, para cada ejecución (dada por Ter_Err o Max_FES).

FES \ n		50	100	200	250	400	500	1000	2000
1e3	1st (Best)	313,59	1157,85	2456,81	2484,74	-	4229,3	4982,45	-
	7st	812,67	1556,83	4211,67	4455,26	-	6278,7	9045,12	-
	13st (Median)	1281,4	2299,87	4843,51	5179,97	-	8662,68	10216,49	-
	19st	1616,19	2772,15	5589,04	6939,94	-	9876,86	11651,89	-
	25st (Worst)	2946,96	5091,8	7449,7	11000,54	-	14498,63	14699,06	-
	Mean	1287,26	2348,54	4974,63	5763,02	-	8335,95	10078,51	-
	Std	644,13	964,97	1328,16	1938,81	-	2660,22	2313,83	-
1e4	1st (Best)	3,49	6,32	9,05	19,48	67,63	65,33	302,22	1088,19
	7st	103,92	31,11	25,82	55,18	95,81	133,77	680,22	2075,02
	13st (Median)	136,28	55,43	43,03	74,95	129,93	199,69	864,49	2606,13
	19st	239,26	128,86	62,05	92,8	169,98	269,87	1110,22	3055,02
	25st (Worst)	782,21	352,8	124,98	163,3	301,15	400,72	1566,57	3630,69
	Mean	186,98	101,94	49,14	74,4	140,62	210,79	914,31	2611,79
	Std	170,01	107,56	29,69	33,6	60,05	94,28	310,41	608,91
1e5 (end)	1st (Best)	2,43	0,44	0,0029	0,0008	0,023	0,084	0,62	3,44
	7st	84,05	17,22	0,2	0,14	0,16	0,29	2,13	7,37
	13st (Median)	127,01	30,25	0,693	0,689	0,4	0,46	3,3	10,3
	19st	188,19	105,61	3,83	3,73	0,93	0,81	4,01	14,3
	25st (Worst)	483,24	295,77	14,32	30,89	7,84	1,79	10,81	27,31
	Mean	151,45	62,79	3,19	3,84	1,37	0,6	3,61	11,46
	Std	117,9	71,33	4,55	7,15	2,04	0,39	2,34	5,55

Tabla F.1: 10.D.**.0005-Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (Function Evaluations) en la finalización, para cada ejecución

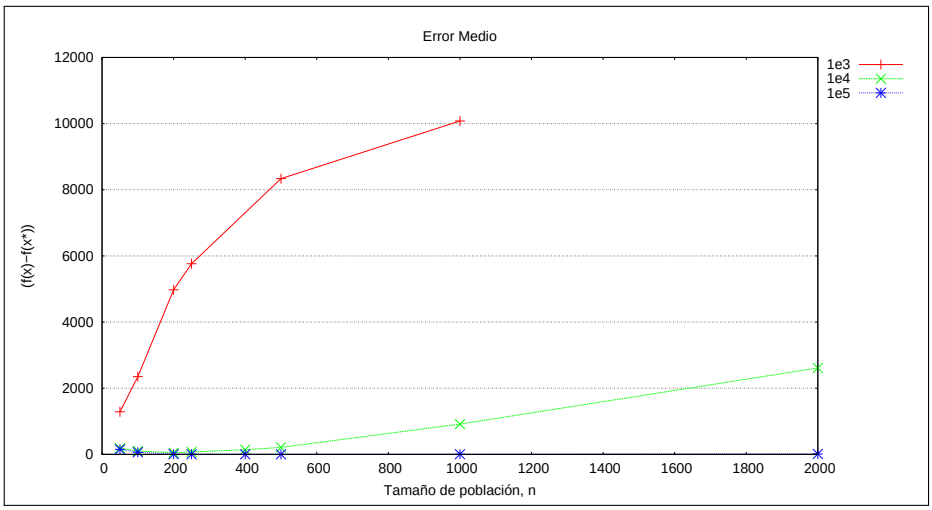

Figura F.1: 10.D.**.0005-Error Medio

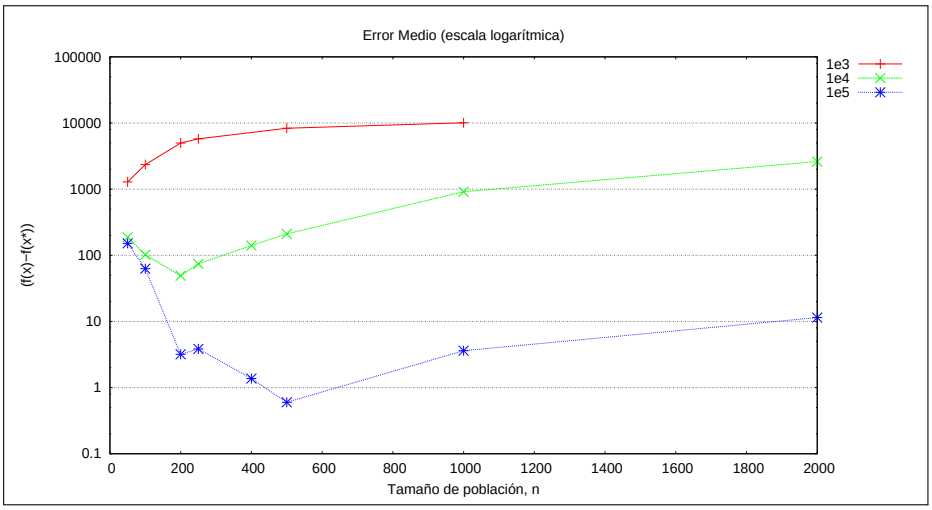

Figura F.2: 10.D.**.0005-Error Medio (escala logarítmica)

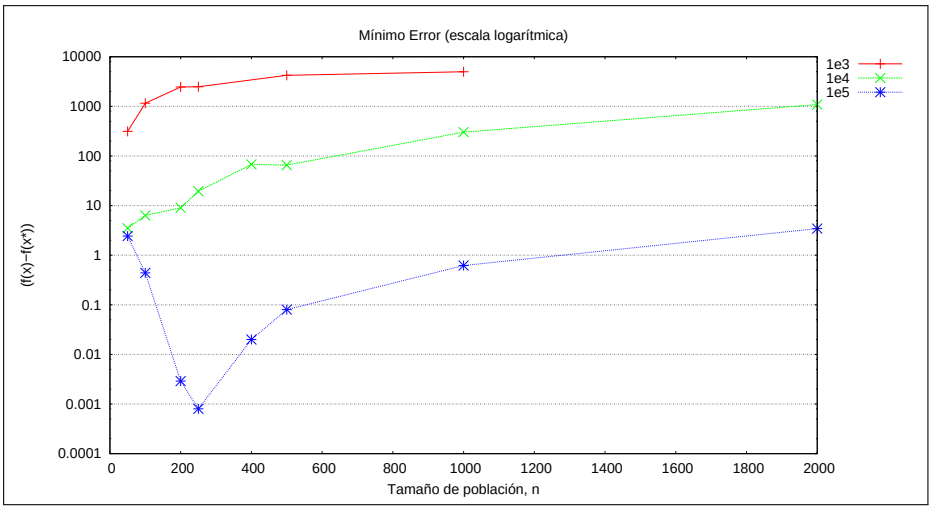

Figura F.3: 10.D.**.0005-Mínimo Error (escala logarítmica)

2. Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el nivel de precisión. Ratio de éxito y rendimiento de éxito.

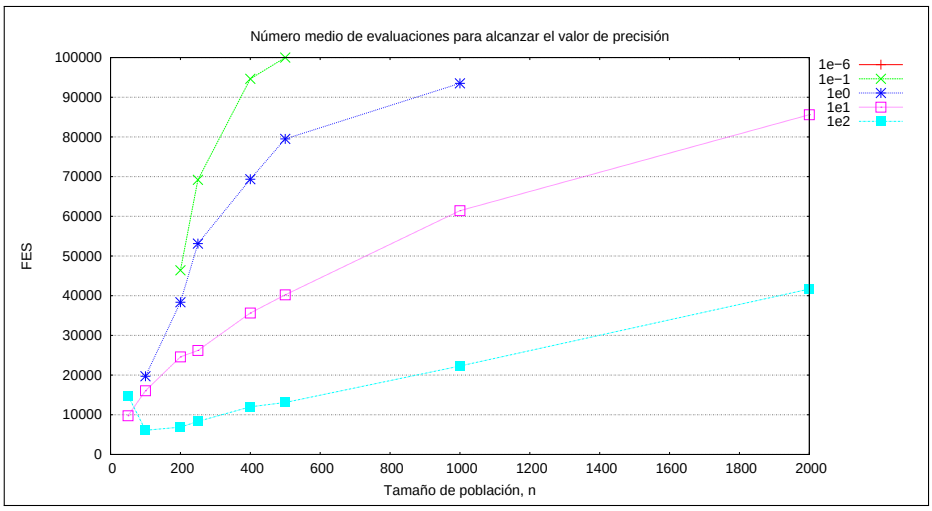

Figura F.4: 10.D.**.0005-Número medio de evaluaciones para alcanzar el valor de precisión

n	1st (Best)	7st	13st (Median)	19st	25st (Worst)	Mean	Std	Nivel de precisión	Success Rate	Success Perform.
50	-	-	-	-	-	-	-	1,00E-006	-	-
100	-	-	-	-	-	-	-	1,00E-006	-	-
200	-	-	-	-	-	-	-	1,00E-006	-	-
250	-	-	-	-	-	-	-	1,00E-006	-	-
400	-	-	-	-	-	-	-	1,00E-006	-	-
500	-	-	-	-	-	-	-	1,00E-006	-	-
1000	-	-	-	-	-	-	-	1,00E-006	-	-
2000	-	-	-	-	-	-	-	1,00E-006	-	-
50	-	-	-	-	-	-	-	1,00E-001	-	-
100	-	-	-	-	-	-	-	1,00E-001	-	-
200	38400	-	-	-	-	46400	11838,92	1,00E-001	0,12	386666,67
250	56250	-	-	-	-	69200	11289,38	1,00E-001	0,2	346000
400	91200	-	-	-	-	94640	2906,54	1,00E-001	0,2	473200
500	100000	-	-	-	-	100000	-	1,00E-001	-	-
1000	-	-	-	-	-	-	-	1,00E-001	-	-
2000	-	-	-	-	-	-	-	1,00E-001	-	-
50	-	-	-	-	-	-	-	1,00E+000	-	-
100	19700	-	-	-	-	19700	0	1,00E+000	0,04	492500
200	19000	36400	54600	-	-	38357,14	12457,73	1,00E+000	0,56	68494,9
250	23750	46500	73750	-	-	53125	18148,36	1,00E+000	0,56	94866,07
400	51200	59200	73200	97200	-	69347,37	15126,85	1,00E+000	0,76	91246,54
500	61000	70000	84500	94500	-	79500	12145,94	1,00E+000	0,88	90340,91
1000	87000	-	-	-	-	93500	9192,39	1,00E+000	0,08	1168750
2000	-	-	-	-	-	-	-	1,00E+000	-	-
50	5900	-	-	-	-	9750	5444,72	1,00E+001	0,08	121875
100	9200	-	-	-	-	16050	7889,87	1,00E+001	0,16	100312,5
200	9400	14800	17800	35000	-	24581,82	19845,3	1,00E+001	0,88	27933,88
250	12500	22250	25000	31250	-	26204,55	10851,99	1,00E+001	0,88	29777,89
400	19200	27600	37200	41200	57600	35632	10201,91	1,00E+001	1	35632
500	29000	36500	39500	45000	50000	40220	6094,87	1,00E+001	1	40220
1000	37000	52000	59000	69000	-	61416,67	15767,1	1,00E+001	0,96	63975,69
2000	66000	94000	-	-	-	85600	10145,61	1,00E+001	0,4	214000
50	1800	13000	-	-	-	14833,33	24565,86	1,00E+002	0,36	41203,7
100	3500	4700	6900			6088,89	2602,91	1,00E+002	0,72	8456,79
200	4200	5800	6200	7600	12600	6880	1882,37	1,00E+002	1	6880
250	5000	6750	8500	9500	12750	8320	2065,99	1,00E+002	1	8320
400	7600	10000	11600	14000	18800	12000	2744,69	1,00E+002	1	12000
500	9500	11000	13000	14500	17000	13100	2313,91	1,00E+002	1	13100
1000	16000	20000	23000	25000	26000	22280	3034,8	1,00E+002	1	22280
2000	32000	38000	42000	44000	50000	41680	4384,82	1,00E+002	1	41680

Tabla F.2: 10.D.**.0005-Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión, ratio de éxito y rendimiento de éxito.

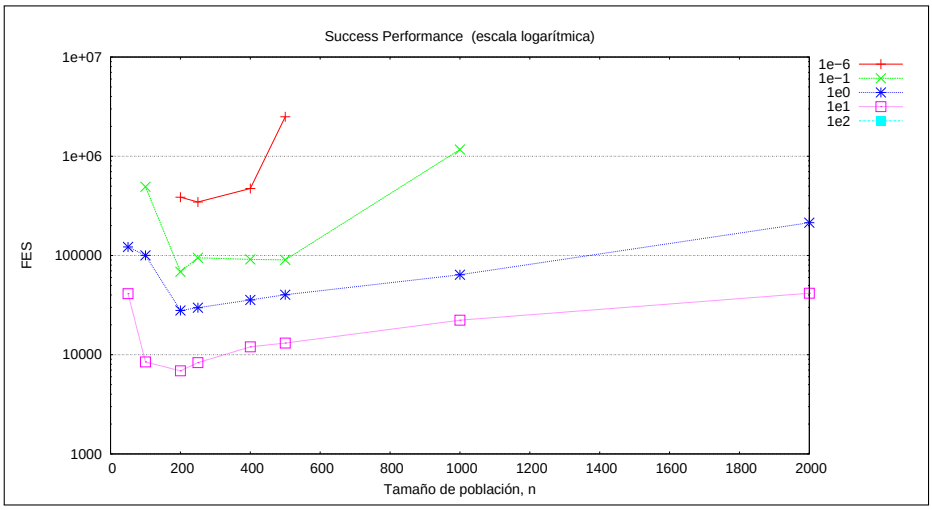

Figura F.5: 10.D.**.0005-Rendimiento de éxito (escala logarítmica)

3. Gráficas de convergencia.

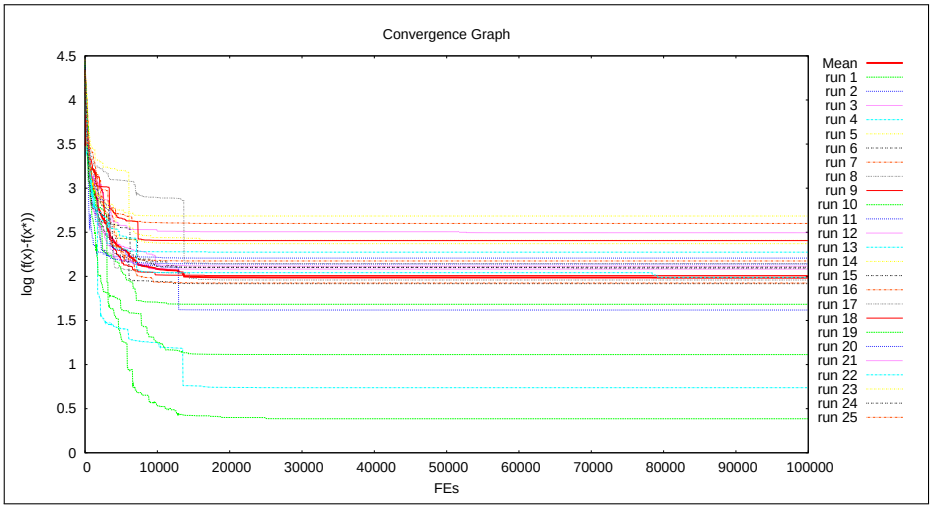

Figura F.6: 10.D.50.0005-Convergencia

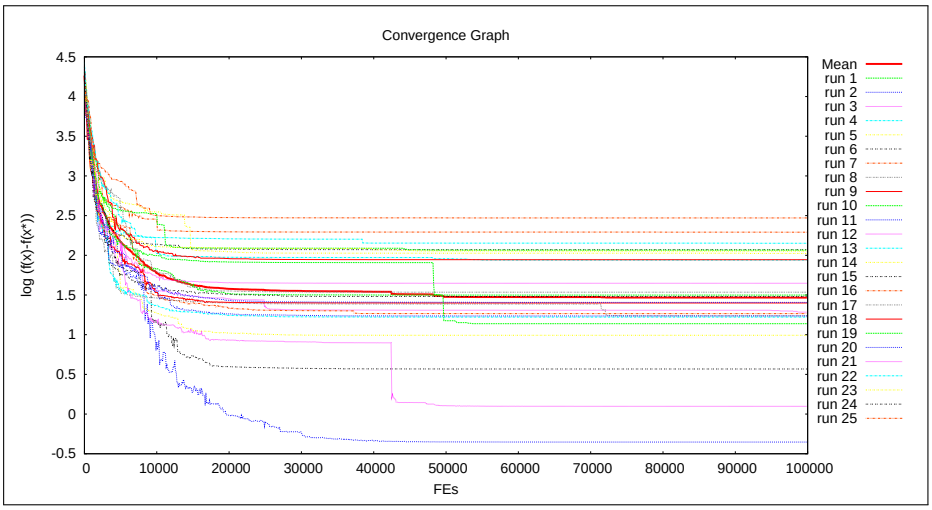

Figura F.7: 10.D.100.0005-Convergencia

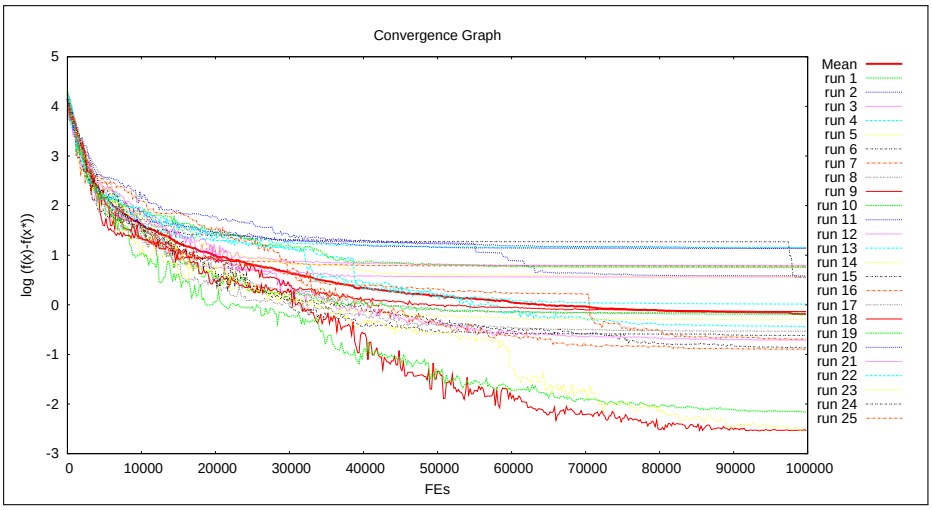

Figura F.8: 10.D.200.0005-Convergencia

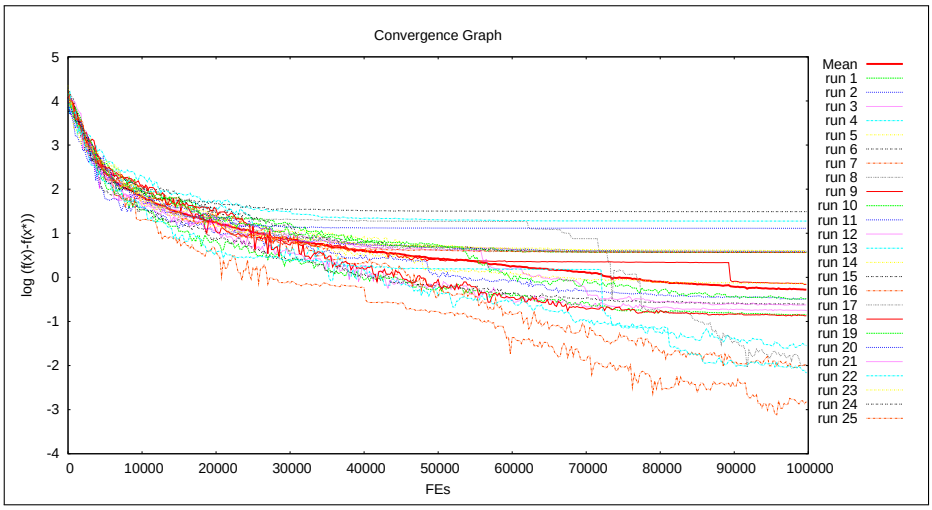

Figura F.9: 10.D.250.0005-Convergencia

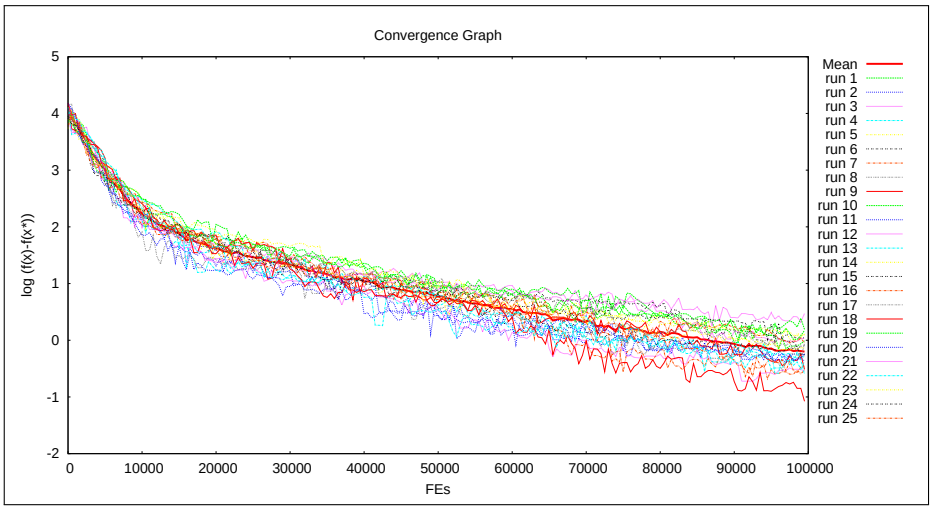

Figura F.10: 10.D.500.0005-Convergencia

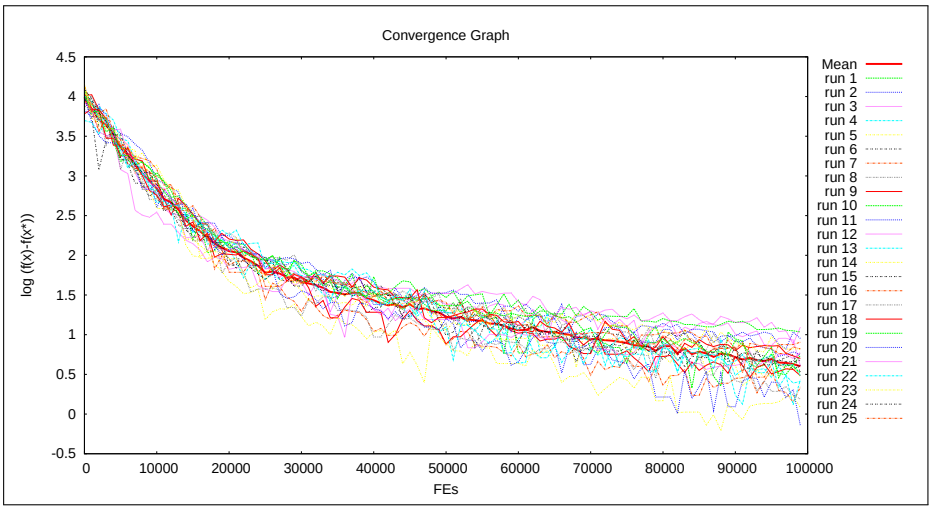

Figura F.11: 10.D.1000.0005-Convergencia

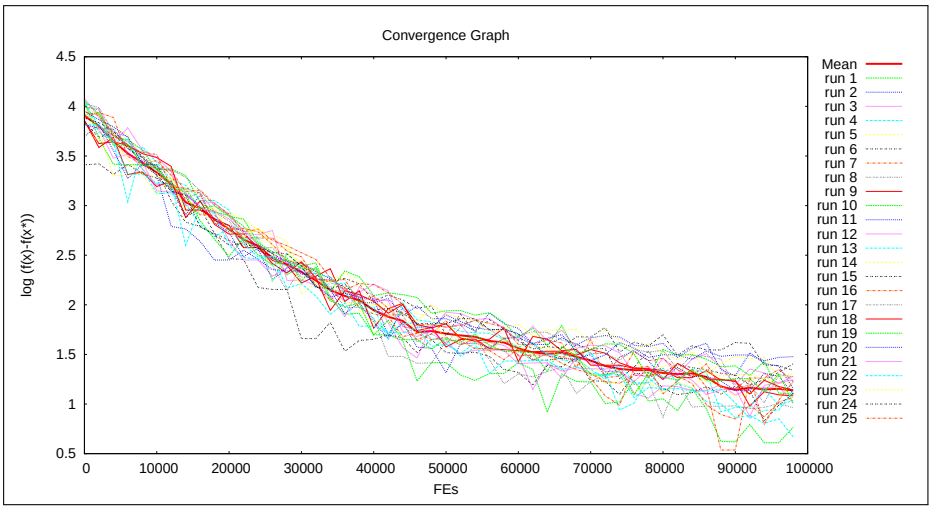

Figura F.12: 10.D.2000.0005-Convergencia

4. Algorithm Complexity.

n	T0	T1	T2'	(T2'-T1)/T0
50	0,18	1,77	21,83	111,43
100			21,34	108,74
200			21,49	109,53
250			21,56	109,97
400			21,49	109,56
500			21,7	110,7
1000			22,1	112,92
2000			23,08	118,38

Tabla F.3: 10.D.**.0005-Algorithm Complexity- computing_time_t2

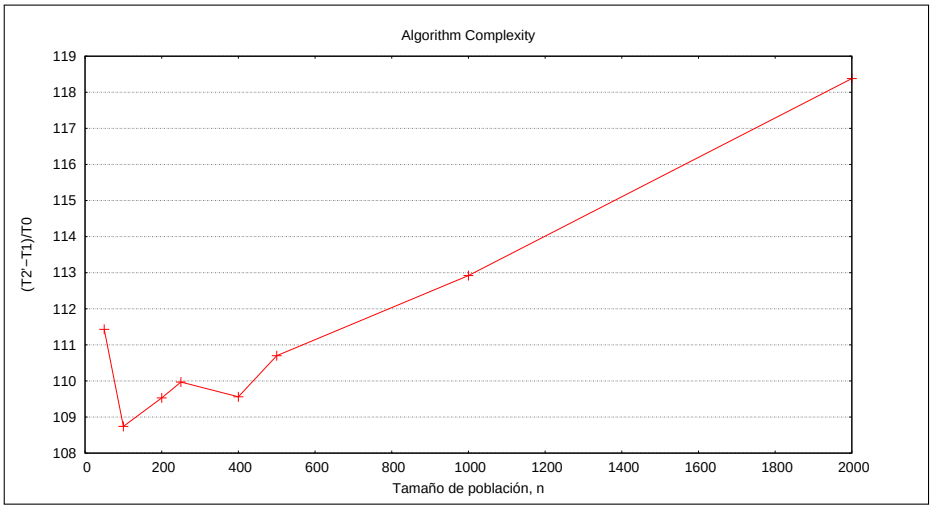

Figura F.13: 10.D.**.0005-Complejidad del algoritmo

F.1.2. Grupo de escenarios 10.R.**.0005

Se trata de los estudios sobre el grupo de escenarios 10.R.**.0005, con el fin de determinar el tamaño de población.

1. Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) en la finalización, para cada ejecución (dada por Ter_Err o Max_FES).

n \ FES		50	100	200	250	400	500	1000	2000
1e3	1st (Best)	960,06	2408,39	3599,25	3681,4		4689,96	6932,96	
	7st	2444,3	3249,61	5551,77	5165,51		7219,77	8886,85	
	13st (Median)	3726,39	3927,26	5993,93	7120,75		8290,12	10184,72	
	19st	4298,98	4767,38	7308,33	7819,68		9213,14	12262,63	
	25st (Worst)	5645,54	6797,89	9929,46	12395,73		13151,39	15155,14	
	Mean	3439,64	4094,38	6467,4	7033,99		8452,98	10415,3	
	Std	1350,88	1172,98	1663,87	2271,91		2324,66	2478,63	
1e4	1st (Best)	30,93	54,51	56,67	104,63	166,31	189,93	816,9	1510,7
	7st	214,35	113,39	139,99	167,9	212,44	377,47	1612,01	2610,63
	13st (Median)	312,84	240,54	187,96	216,4	299,58	453,44	1846,98	3454,2
	19st	401,47	341,05	310,24	270,44	354,17	563,67	1981,84	4020,04
	25st (Worst)	2414,14	1038,89	791,95	500,65	577,21	1059,77	3591,87	6417,94
	Mean	489,9	321,6	237,71	232,2	306,3	482,26	1888,03	3411,04
	Std	599,38	292,25	156,54	99,02	112,42	181,28	628,92	1127,97
1e5 (end)	1st (Best)	19,72	7,3	14,7	8,41	6,26	7,85	9,22	10,27
	7st	125,7	39,5	22,84	20,22	11,52	12,01	16,12	28,59
	13st (Median)	211,5	104,82	42,94	32,23	21,81	21,11	20,68	33,96
	19st	296,54	214,18	66,06	46,48	35,43	30,97	27,02	41,14
	25st (Worst)	2275,47	935,41	204,08	153,28	83,09	61,7	51,82	71,62
	Mean	368,59	186,54	56,97	47,17	26,93	24,04	22,82	37,49
	Std	573,05	246,19	48,28	43,29	19,55	14,62	10,36	14,67

Tabla F.4: 10.R.**.0005-Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) en la finalización, para cada ejecución

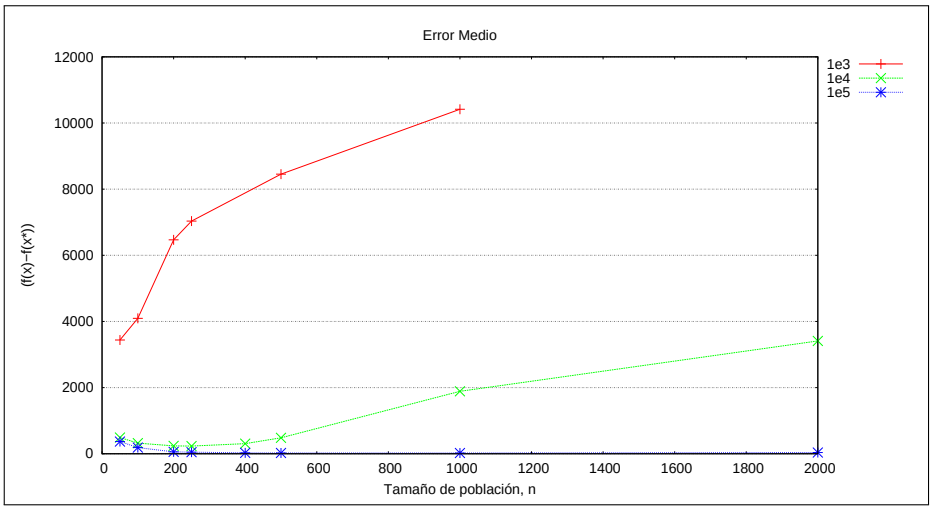

Figura F.14: 10.R.**.0005-Error Medio

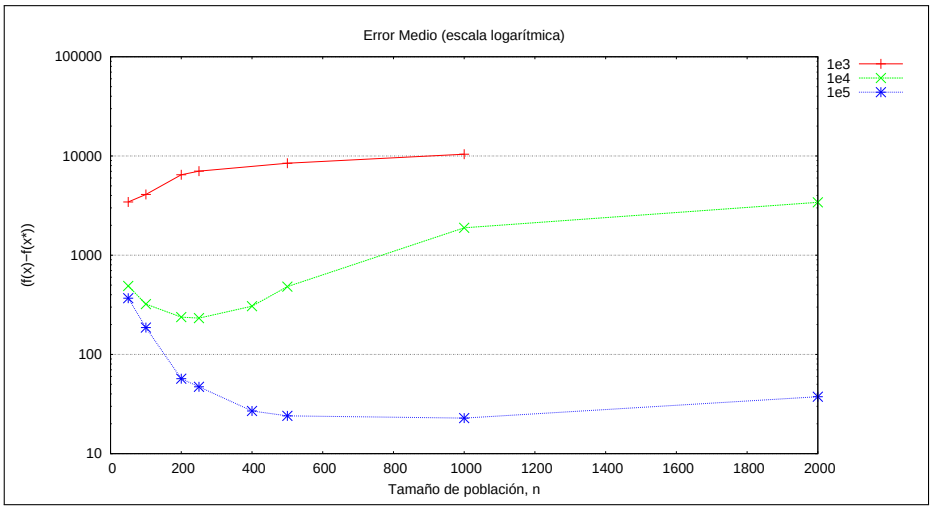

Figura F.15: 10.R.**.0005-Error Medio (escala logarítmica)

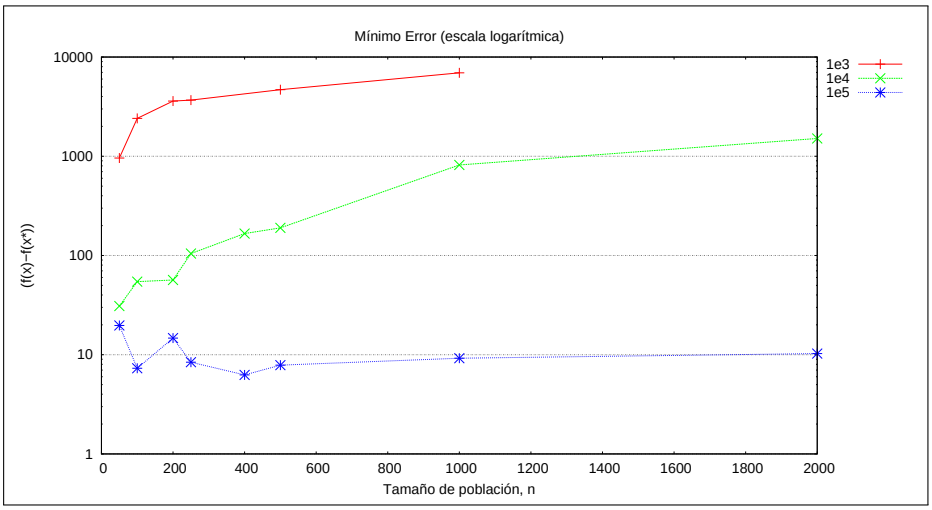

Figura F.16: 10.R.**.0005-Mínimo Error (escala logarítmica)

2. Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el nivel de precisión. Ratio de éxito y rendimiento de éxito.

n	1st (Best)	7st	13st (Median)	19st	25st (Worst)	Mean	Std	Nivel de precisión	Success Rate	Success Perform.
50	-	-	-	-	-	-	-	1,00E-006	-	-
100	-	-	-	-	-	-	-	1,00E-006	-	-
200	-	-	-	-	-	-	-	1,00E-006	-	-
250	-	-	-	-	-	-	-	1,00E-006	-	-
400	-	-	-	-	-	-	-	1,00E-006	-	-
500	-	-	-	-	-	-	-	1,00E-006	-	-
1000	-	-	-	-	-	-	-	1,00E-006	-	-
2000	-	-	-	-	-	-	-	1,00E-006	-	-
50	-	-	-	-	-	-	-	1,00E-001	-	-
100	-	-	-	-	-	-	-	1,00E-001	-	-
200	-	-	-	-	-	-	-	1,00E-001	-	-
250	-	-	-	-	-	-	-	1,00E-001	-	-
400	-	-	-	-	-	-	-	1,00E-001	-	-
500	-	-	-	-	-	-	-	1,00E-001	-	-
1000	-	-	-	-	-	-	-	1,00E-001	-	-
2000	-	-	-	-	-	-	-	1,00E-001	-	-
50	-	-	-	-	-	-	-	1,00E+000	-	-
100	-	-	-	-	-	-	-	1,00E+000	-	-
200	-	-	-	-	-	-	-	1,00E+000	-	-
250	-	-	-	-	-	-	-	1,00E+000	-	-
400	-	-	-	-	-	-	-	1,00E+000	-	-
500	-	-	-	-	-	-	-	1,00E+000	-	-
1000	-	-	-	-	-	-	-	1,00E+000	-	-
2000	-	-	-	-	-	-	-	1,00E+000	-	-
50	-	-	-	-	-	-	-	1,00E+001	-	-
100	54200	-	-	-	-	63950	13788,58	1,00E+001	0,08	799375
200	-	-	-	-	-	-	-	1,00E+001	-	-
250	99000	-	-	-	-	99000		1,00E+001	0,04	2475000
400	55600	-	-	-	-	70560	11550,24	1,00E+001	0,2	352800
500	47500	-	-	-	-	73666,67	25299,87	1,00E+001	0,12	613888,89
1000	75000	-	-	-	-	75000		1,00E+001	0,04	1875000
2000	-	-	-	-	-	-	-	1,00E+001	-	-
50	5600	-	-	-	-	14112,5	11591,26	1,00E+002	0,16	88203,13
100	5800	10700				15475	16758,99	1,00E+002	0,48	32239,58
200	7800	14200	21200	25200		17866,67	6282,46	1,00E+002	0,84	21269,84
250	10250	13500	23250	33750		22285,71	12006,32	1,00E+002	0,84	26530,61
400	12000	15600	17200	27600	36400	20768	7797,37	1,00E+002	1	20768
500	14000	18500	21500	26500	38000	23100	6301,45	1,00E+002	1	23100
1000	21000	33000	37000	39000	67000	37120	8847,41	1,00E+002	1	37120
2000	46000	56000	62000	64000	78000	60560	9138,2	1,00E+002	1	60560

Tabla F.5: 10.R.**.0005-Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión, ratio de éxito y rendimiento de éxito.

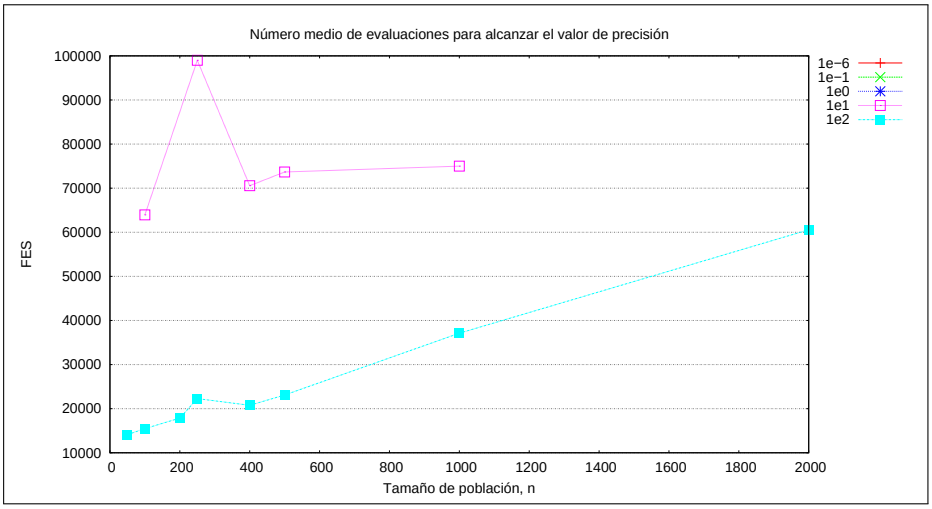

Figura F.17: 10.R.**.0005-Número medio de evaluaciones para alcanzar el valor de precisión

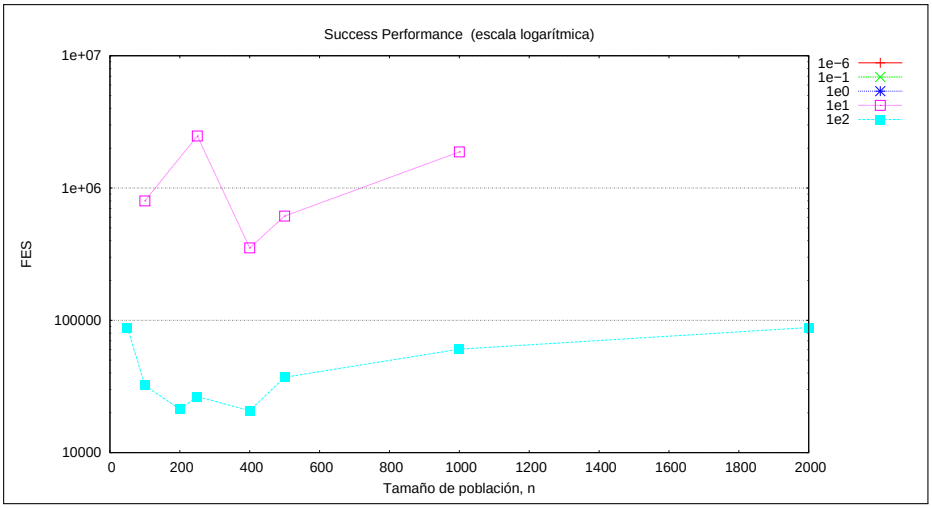

Figura F.18: 10.R.**.0005-Rendimiento de éxito (escala logarítmica)

3. Gráficas de convergencia.

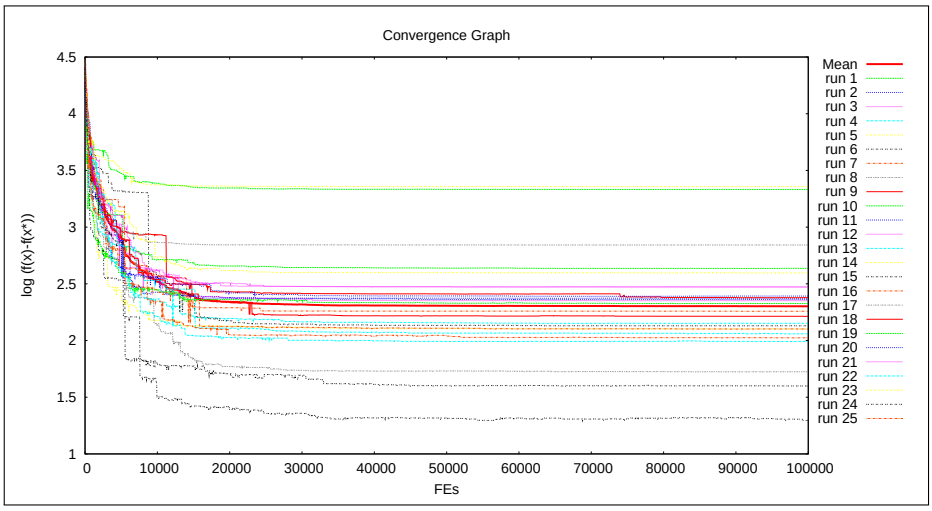

Figura F.19: 10.R.50.0005-Convergencia

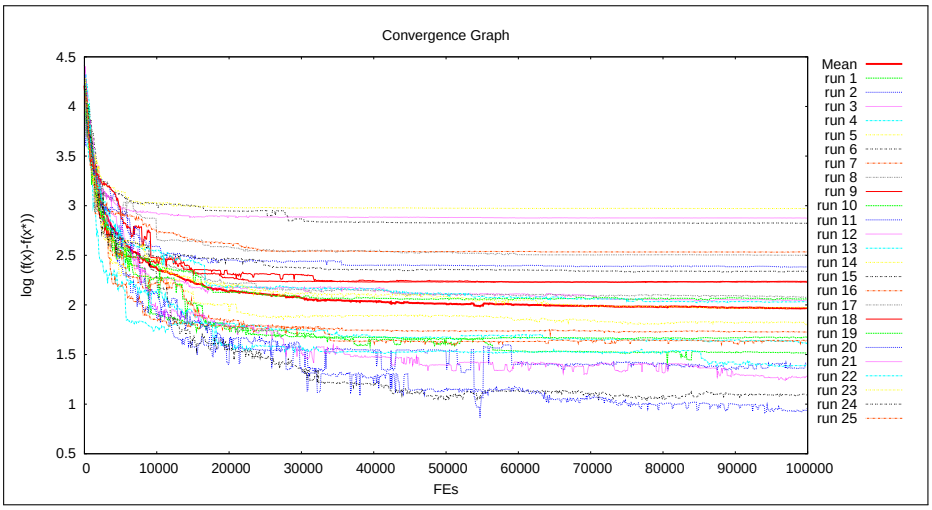

Figura F.20: 10.R.100.0005-Convergencia

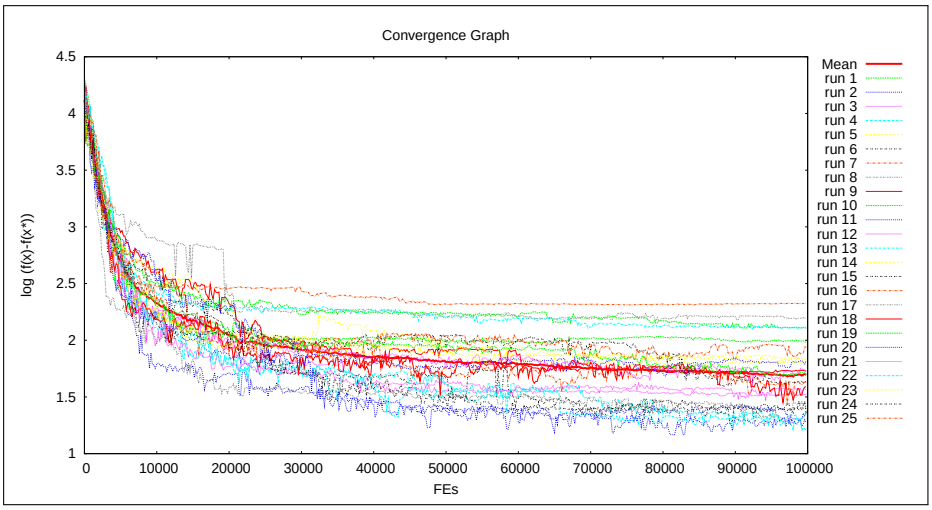

Figura F.21: 10.R.200.0005-Convergencia

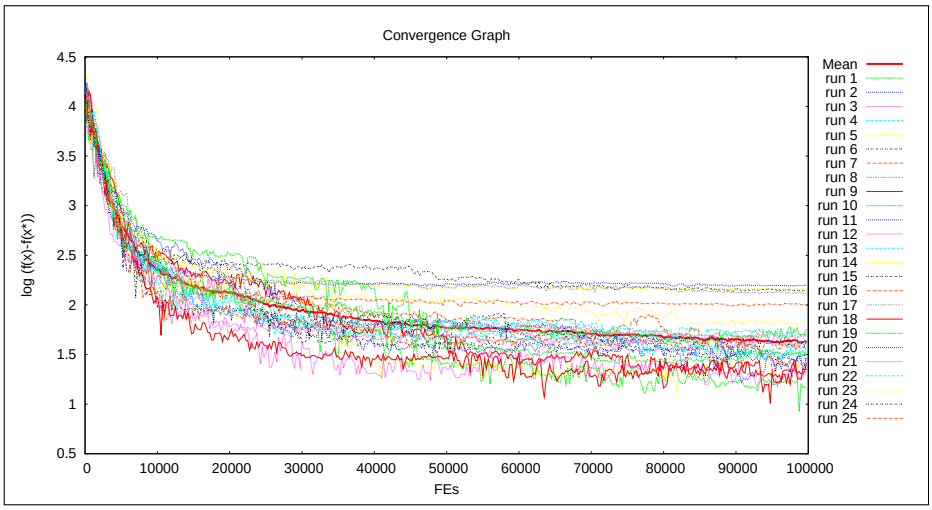

Figura F.22: 10.R.250.0005-Convergencia

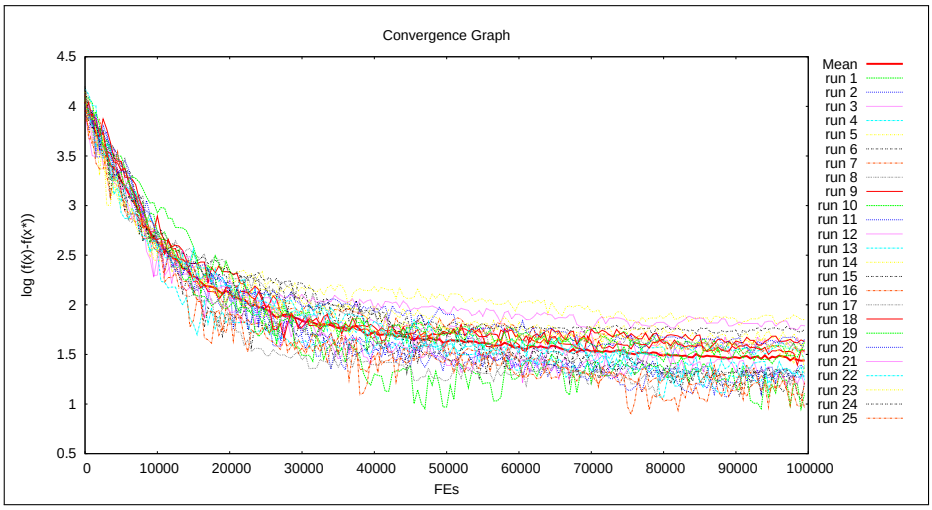

Figura F.23: 10.R.500.0005-Convergencia

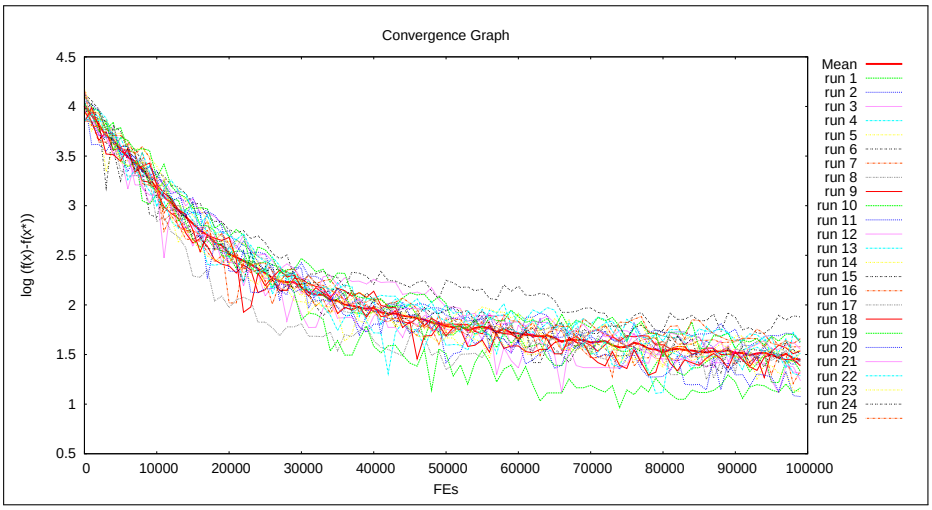

Figura F.24: 10.R.1000.0005-Convergencia

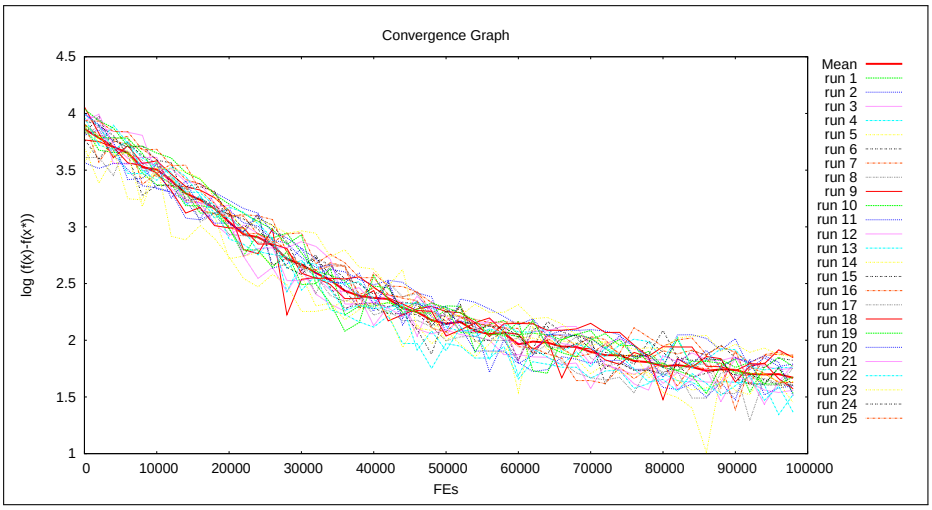

Figura F.25: 10.R.2000.0005-Convergencia

4. Algorithm Complexity.

n	T0	T1	T2'	(T2'-T1)/T0
50	0,18	1,77	21,41	109,1
100			21,51	109,64
200			21,36	108,81
250			21,25	108,21
400			21,42	109,15
500			21,36	108,27
1000			21,35	108,79
2000			22,39	114,55

Tabla F.6: 10.R.**.0005-Algorithm Complexity- computing_time_t2

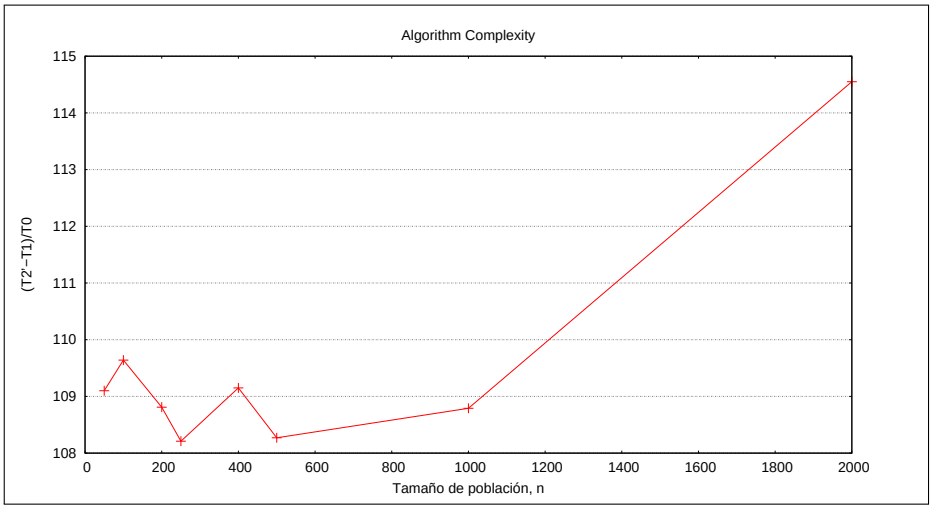

Figura F.26: 10.R.**.0005-Complejidad del algoritmo

F.1.3. Grupo de escenarios 50.D.**.0005

Se trata de los estudios sobre el grupo de escenarios 50.D.**.0005, con el fin de determinar el tamaño de población.

1. Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) y en la finalización ($5 \cdot 10^5$), para cada ejecución (dada por Ter_Err o Max_FES).

FES \ n		250	400	500	1000	2000	2500
1e3	1st (Best)	116538,21	-	135992,26	159220,5	-	-
	7st	133994,05	-	156653,8	168188,05	-	-
	13st (Median)	142560,55	-	164726,34	175322,94	-	-
	19st	152216,3	-	167674,48	183275,55	-	-
	25st (Worst)	182075,86	-	177201,99	194692,33	-	-
	Mean	142801,87	-	161660,12	175535,24	-	-
	Std	15102,9	-	9310,89	9861,44	-	-
1e4	1st (Best)	9585,79	13853,19	18309,68	54912,22	89129,18	90598,42
	7st	12173,62	20471,46	27605,16	60542,13	101062,09	110995,14
	13st (Median)	13042,74	21722,05	30101,48	65509,75	104813,37	118484,88
	19st	13930,03	23919,78	34233,65	69691,22	108585,08	124122,67
	25st (Worst)	19590,28	29133,2	37877,47	76333,75	118493,3	140301,83
	Mean	13321,38	21884,41	30551,02	65597,07	104935,94	116592,88
	Std	1845,27	3663,5	4768,8	5935,8	7464,76	11725,44
1e5	1st (Best)	373,21	402,25	555,57	896,09	2523,23	4248,51
	7st	620,71	770,82	780,13	1368,07	2828,41	4787,65
	13st (Median)	1021,17	1027,9	839,44	1506,62	3175,88	5224,15
	19st	1606,32	1234,94	1088,54	1636,05	3722,26	5573,44
	25st (Worst)	4308,87	1703,21	1588,77	2311,42	4043,38	6691,13
	Mean	1277,85	1031,12	942,14	1505,99	3225,56	5223,55
	Std	872,38	343,59	295,37	339,14	457,49	673,55
5e5 (end)	1st (Best)	234,16	167,66	39,96	38,64	39,82	105,8
	7st	404,37	384,82	247,8	114,02	79,81	141,55
	13st (Median)	596,58	504,56	366,36	199,18	114,61	175,41
	19st	1083,27	759,97	531,23	258,57	128,49	225,56
	25st (Worst)	3818,94	1411,68	1118,3	724,92	518,87	368,51
	Mean	907,67	611,47	427,32	209,55	127,09	195,77
	Std	836,35	319,79	297,14	138,91	95,21	70,11

Tabla F.7: 50.D.**.0005-Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) y en la finalización ($5 \cdot 10^5$), para cada ejecución

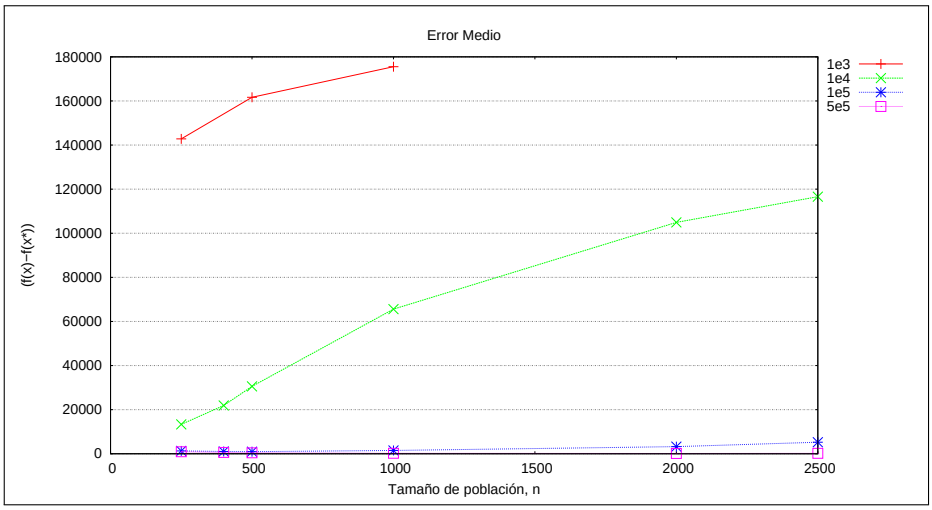

Figura F.27: 50.D.**.0005-Error Medio

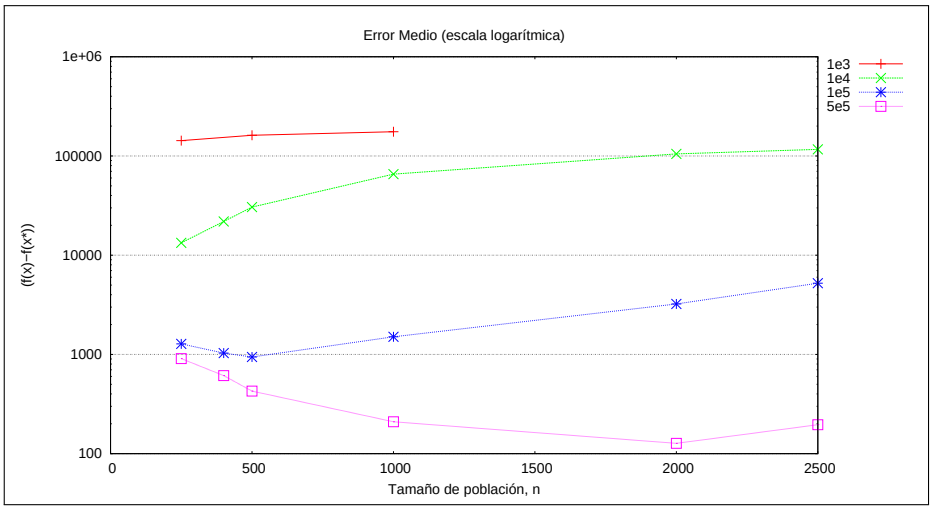

Figura F.28: 50.D.**.0005-Error Medio (escala logarítmica)

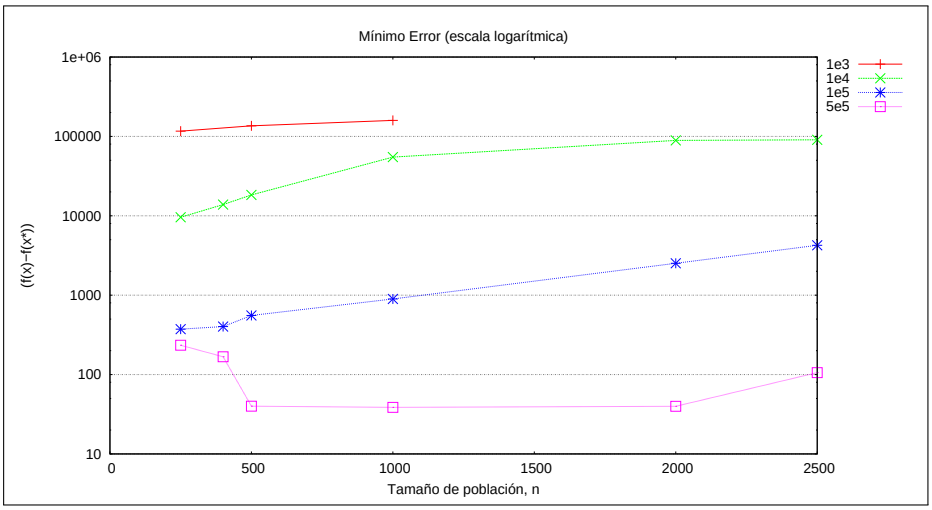

Figura F.29: 50.D.**.0005-Mínimo Error (escala logarítmica)

2. Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el nivel de precisión. Ratio de éxito y rendimiento de éxito.

n	1st (Best)	7st	13st (Median)	19st	25st (Worst)	Mean	Std	Nivel de precisión	Success Rate	Success Perform.
250	-	-	-	-	-	-	-	1,00E-006	-	-
400	-	-	-	-	-	-	-	1,00E-006	-	-
500	-	-	-	-	-	-	-	1,00E-006	-	-
1000	-	-	-	-	-	-	-	1,00E-006	-	-
2000	-	-	-	-	-	-	-	1,00E-006	-	-
2500	-	-	-	-	-	-	-	1,00E-006	-	-
250	-	-	-	-	-	-	-	1,00E+000	-	-
400	-	-	-	-	-	-	-	1,00E+000	-	-
500	-	-	-	-	-	-	-	1,00E+000	-	-
1000	-	-	-	-	-	-	-	1,00E+000	-	-
2000	-	-	-	-	-	-	-	1,00E+000	-	-
2500	-	-	-	-	-	-	-	1,00E+000	-	-
250	-	-	-	-	-	-	-	1,00E+001	-	-
400	-	-	-	-	-	-	-	1,00E+001	-	-
500	-	-	-	-	-	-	-	1,00E+001	-	-
1000	-	-	-	-	-	-	-	1,00E+001	-	-
2000	-	-	-	-	-	-	-	1,00E+001	-	-
2500	-	-	-	-	-	-	-	1,00E+001	-	-
250	-	-	-	-	-	-	-	1,00E+002	-	-
400	-	-	-	-	-	-	-	1,00E+002	-	-
500	338500	-	-	-	-	405250	94398,76	1,00E+002	0,08	5065625
1000	279000	-	-	-	-	371000	82779,83	1,00E+002	0,2	1855000
2000	354000	454000	-	-	-	437555,56	40072,16	1,00E+002	0,36	1215432,1
2500	-	-	-	-	-	-	-	1,00E+002	-	-
250	44000	66500	105750	-	-	106819,44	77351,56	1,00E+003	0,72	148360,34
400	56000	81200	104400	157200	-	109636,36	41471,95	1,00E+003	0,88	124586,78
500	67000	83000	87000	110000	-	94340,91	29088,69	1,00E+003	0,88	107205,58
1000	91000	116000	130000	140000	228000	130360	26573,61	1,00E+003	1	130360
2000	162000	170000	180000	190000	234000	183280	18183,14	1,00E+003	1	183280
2500	177500	220000	235000	247500	277500	232600	23853,72	1,00E+003	1	232600

Tabla F.8: 50.D.**.0005-Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión, ratio de éxito y rendimiento de éxito.

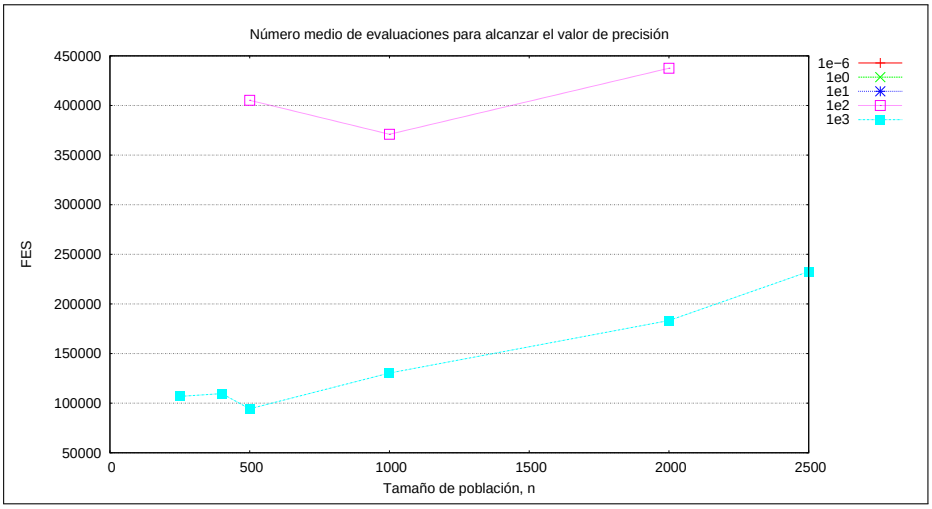

Figura F.30: 50.D.**.0005-Número medio de evaluaciones para alcanzar el valor de precisión

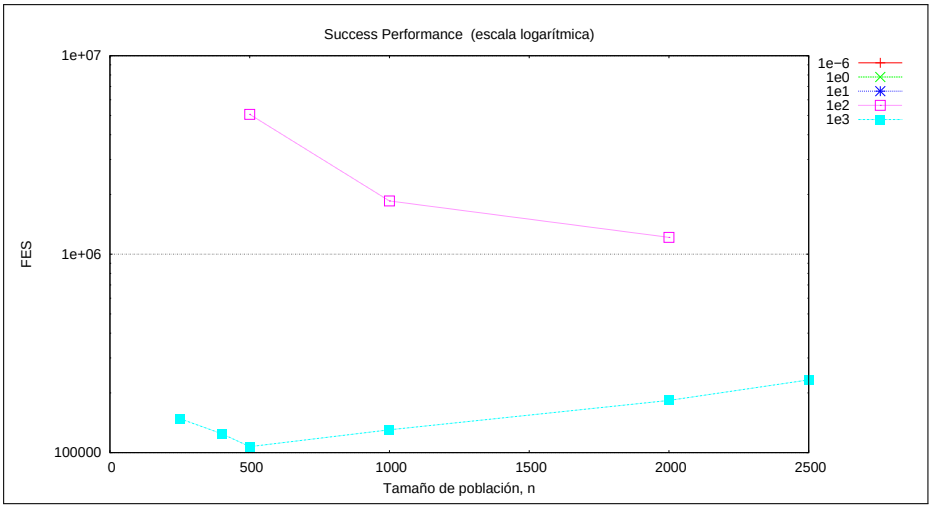

Figura F.31: 50.D.**.0005-Rendimiento de éxito (escala logarítmica)

3. Gráficas de convergencia.

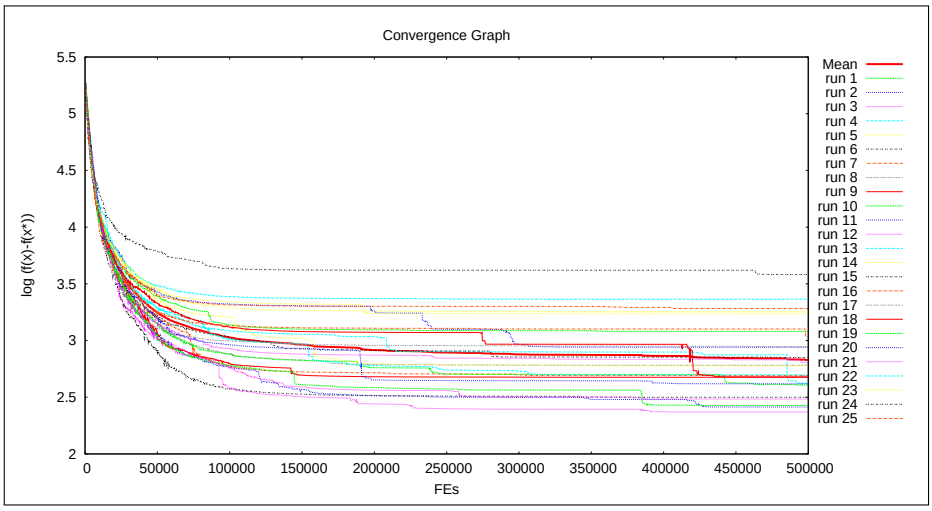

Figura F.32: 50.D.250.0005-Convergencia

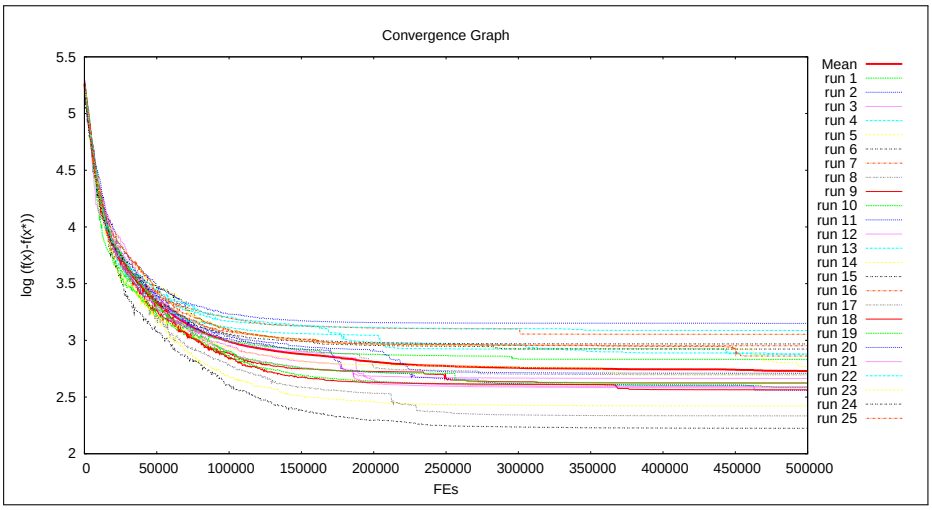

Figura F.33: 50.D.400.0005-Convergencia

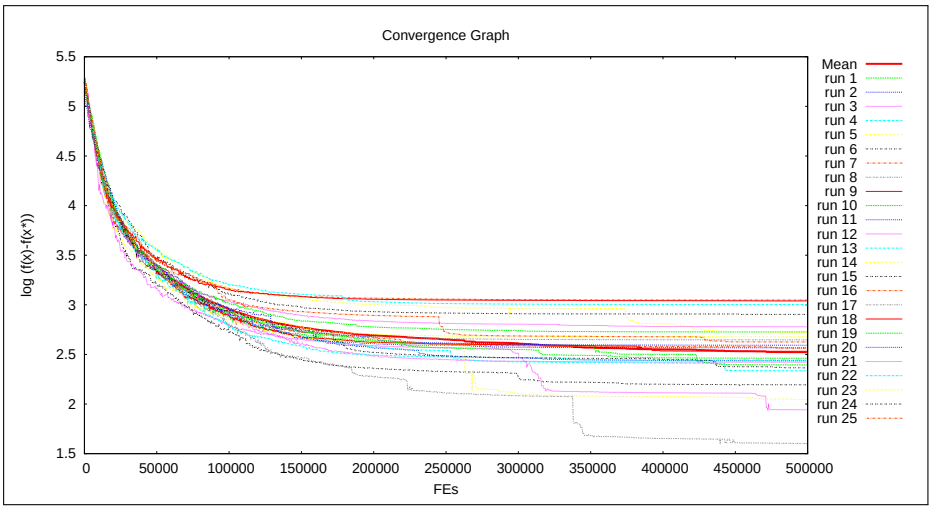

Figura F.34: 50.D.500.0005-Convergencia

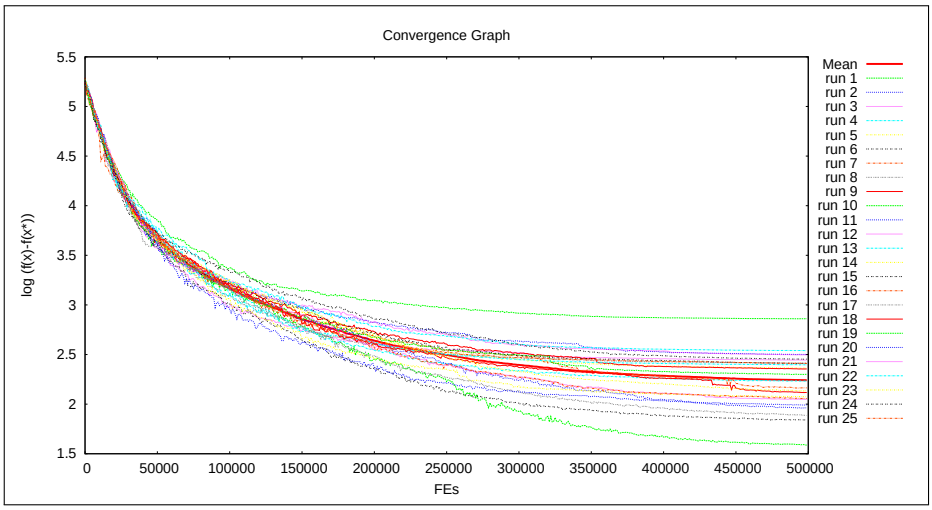

Figura F.35: 50.D.1000.0005-Convergencia

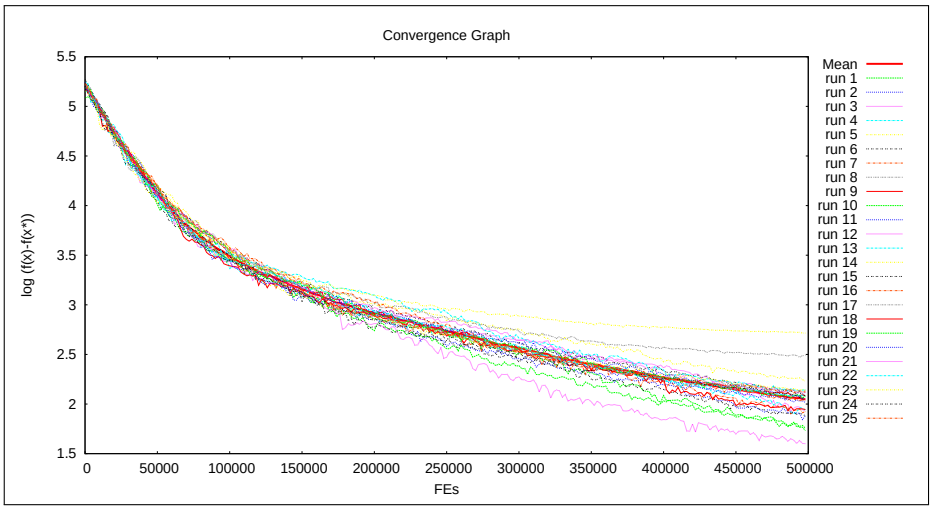

Figura F.36: 50.D.2000.0005-Convergencia

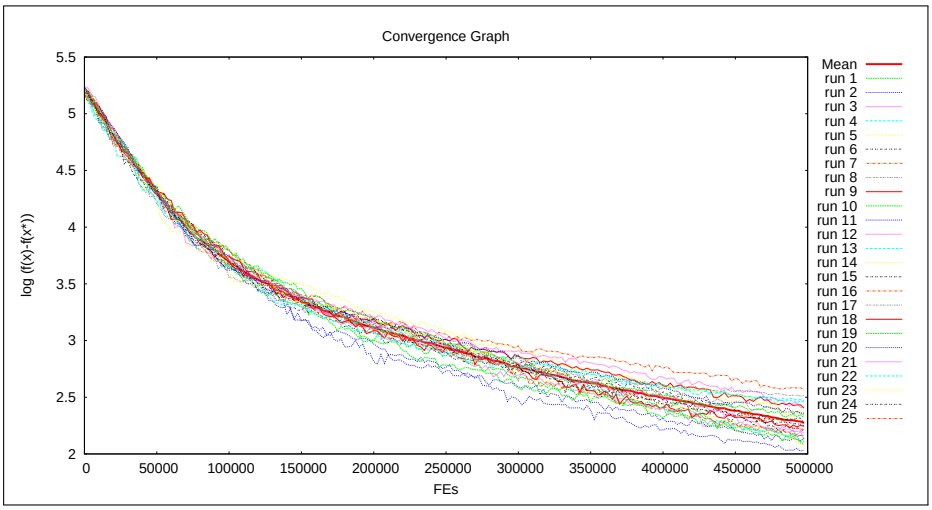

Figura F.37: 50.D.2500.0005-Convergencia

4. Algorithm Complexity.

n	T0	T1	T2'	(T2'-T1)/T0
250	0,18	1,77	494,49	2737,31
400			498,06	2757,14
500			498,76	2761,05
1000			500,62	2771,4
2000			507,66	2810,48
2500			511,59	2832,31

Tabla F.9: 50.D.**.0005-Algorithm Complexity- computing_time_t2

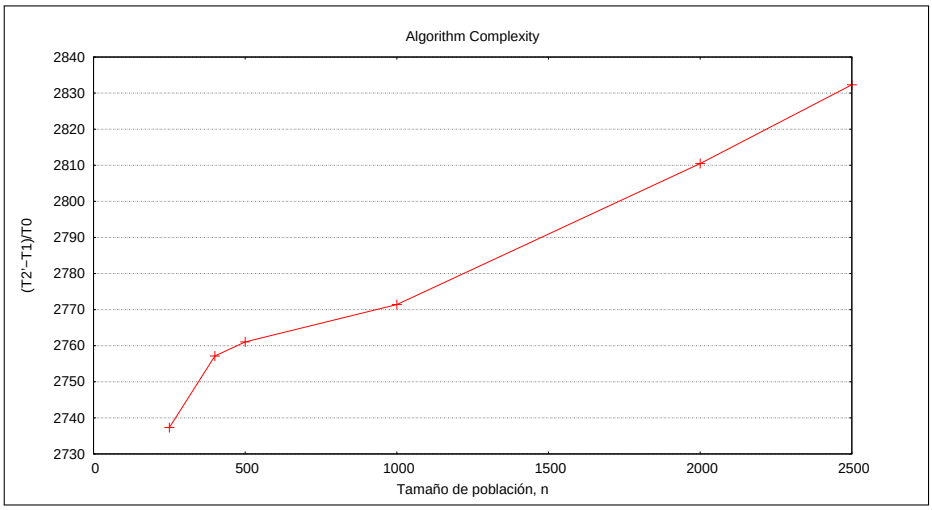

Figura F.38: 50.D.**.0005-Complejidad del algoritmo

F.1.4. Grupo de escenarios 50.R.**.0005

Se trata de los estudios sobre el grupo de escenarios 50.R.**.0005, con el fin de determinar el tamaño de población.

1. Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) y en la finalización ($5 \cdot 10^5$), para cada ejecución (dada por Ter_Err o Max_FES).

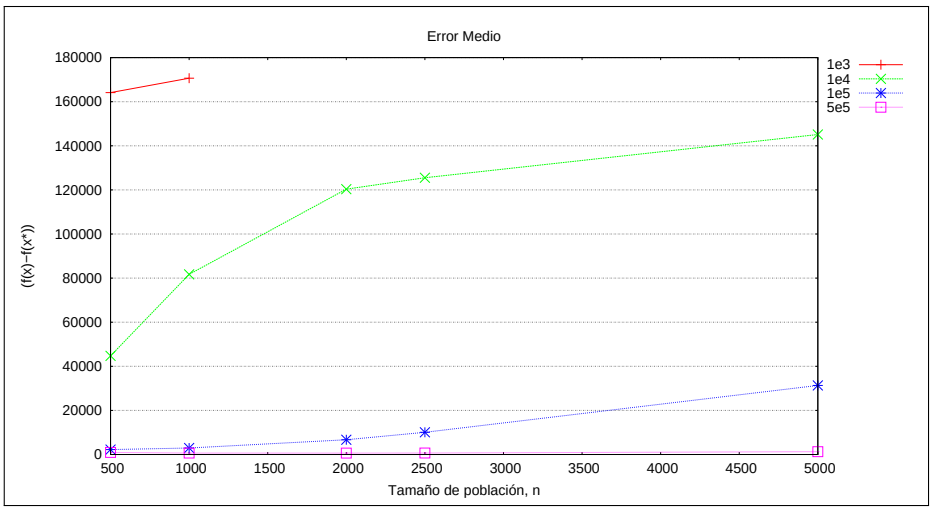

Figura F.39: 50.R.**.0005-Error Medio

FES \ n		500	1000	2000	2500	5000
1e3	1st (Best)	143164,37	127885,79	-	-	-
	7st	153635,05	166456,67	-	-	-
	13st (Median)	160347,56	173320,02	-	-	-
	19st	173456,87	179391,58	-	-	-
	25st (Worst)	190023,72	196269,89	-	-	-
	Mean	164117,24	170657,52	-	-	-
	Std	13623,13	15526,14	-	-	-
1e4	1st (Best)	37725,39	62594,12	1,01E+005	102248,99	122071,45
	7st	40548,56	77651,02	1,14E+005	120069,89	140584,5
	13st (Median)	44391,96	82893,69	120864,17	125880,63	145995,33
	19st	47842,06	86775,58	125743,76	133683,27	149500,11
	25st (Worst)	54698,06	97233,86	132915,85	136364,72	166531,63
	Mean	44729,48	81705,04	1,20E+005	125519,17	145126,46
	Std	4691,23	8955,93	8,63E+003	9118,29	9079,41
1e5	1st (Best)	1556,47	2116,33	5,47E+003	6224,1	25321,96
	7st	1771,09	2417,13	6,09E+003	9612,56	28983,15
	13st (Median)	2266,36	2841,59	6,60E+003	10069,48	31247,07
	19st	2512,44	3185,13	7,07E+003	10571,99	32667,69
	25st (Worst)	3105,4	5076,79	8679,6	14757,48	37870,84
	Mean	2229,24	2950,49	6702,26	10072,93	31315,64
	Std	483,49	725,74	7,77E+002	1636,31	3022,12
5e5 (end)	1st (Best)	549,19	260,46	4,50E+002	405,02	996,23
	7st	681,88	496,18	5,89E+002	640,4	1194,81
	13st (Median)	812,99	562,3	6,41E+002	708,51	1271,99
	19st	1095,32	705,4	6,79E+002	777,9	1387,54
	25st (Worst)	2005,58	1896,08	8,28E+002	835,89	1659,12
	Mean	941	658,48	636,05	687,16	1292,96
	Std	384,15	329,5	92,28	119,63	163,56

Tabla F.10: 50.R.**.0005-Valor de error de la función $(f(x) - f(x^*))$ después de $1 \cdot 10^3$, $1 \cdot 10^4$ y $1 \cdot 10^5$ FES (evaluación la función de salud) y en la finalización ($5 \cdot 10^5$), para cada ejecución

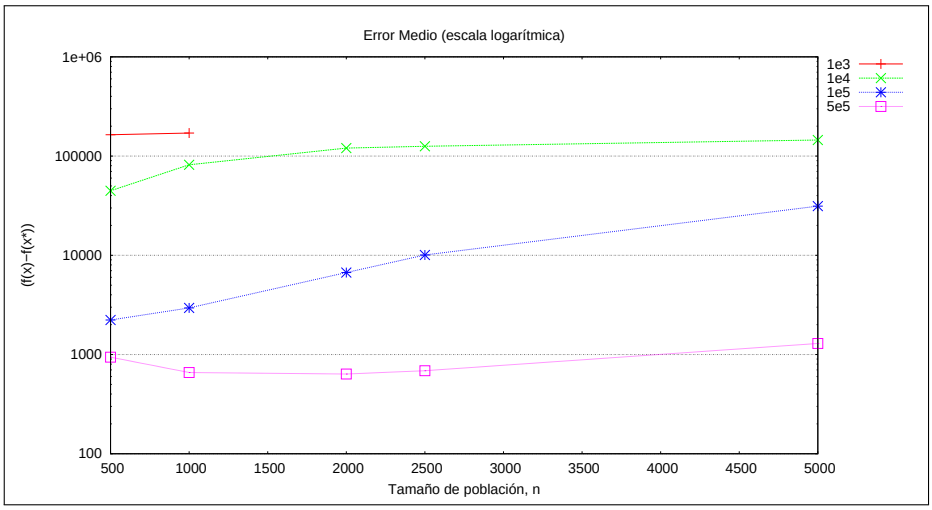

Figura F.40: 50.R.**.0005-Error Medio (escala logarítmica)

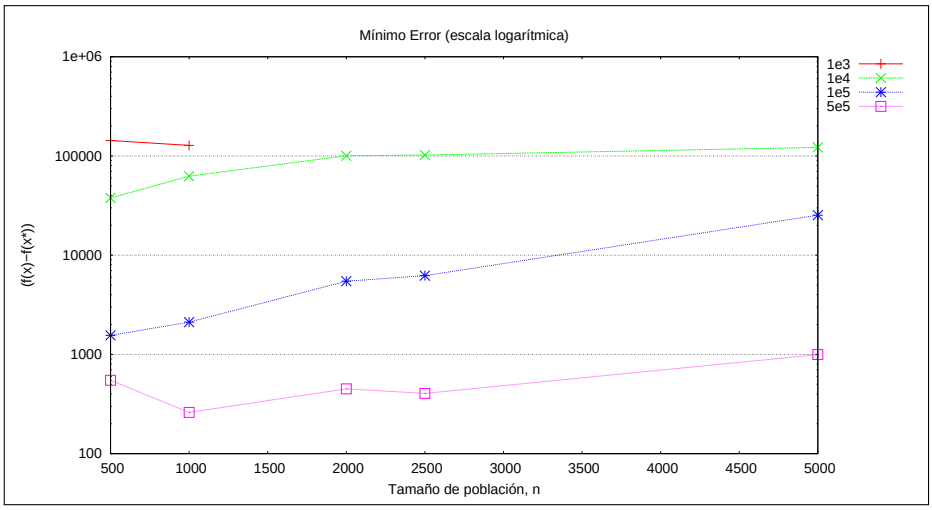

Figura F.41: 50.R.**.0005-Mínimo Error (escala logarítmica)

2. Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el nivel de precisión. Ratio de éxito y rendimiento de éxito.

n	1st (Best)	7st	13st (Median)	19st	25st (Worst)	Mean	Std	Nivel de precisión	Success Rate	Success Perform.
500	-	-	-	-	-	-	-	1,00E-006	-	-
1000	-	-	-	-	-	-	-	1,00E-006	-	-
2000	-	-	-	-	-	-	-	1,00E-006	-	-
2500	-	-	-	-	-	-	-	1,00E-006	-	-
5000	-	-	-	-	-	-	-	1,00E-006	-	-
500	-	-	-	-	-	-	-	1,00E+000	-	-
1000	-	-	-	-	-	-	-	1,00E+000	-	-
2000	-	-	-	-	-	-	-	1,00E+000	-	-
2500	-	-	-	-	-	-	-	1,00E+000	-	-
5000	-	-	-	-	-	-	-	1,00E+000	-	-
500	-	-	-	-	-	-	-	1,00E+001	-	-
1000	-	-	-	-	-	-	-	1,00E+001	-	-
2000	-	-	-	-	-	-	-	1,00E+001	-	-
2500	-	-	-	-	-	-	-	1,00E+001	-	-
5000	-	-	-	-	-	-	-	1,00E+001	-	-
500	-	-	-	-	-	-	-	1,00E+002	-	-
1000	-	-	-	-	-	-	-	1,00E+002	-	-
2000	-	-	-	-	-	-	-	1,00E+002	-	-
2500	-	-	-	-	-	-	-	1,00E+002	-	-
5000	-	-	-	-	-	-	-	1,00E+002	-	-
500	165500	204500	335000	-	-	249625	77734,06	1,00E+003	0,64	390039,06
1000	167000	218000	247000	304000	-	268521,74	80916,27	1,00E+003	0,92	291871,46
2000	270000	302000	320000	348000	434000	327200	39174,82	1,00E+003	1	327200
2500	267500	345000	365000	397500	437500	365700	44214,91	1,00E+003	1	365700
5000	495000	-	-	-	-	495000	-	1,00E+003	0,04	12375000

Tabla F.11: 50.R.**.0005-Número máximo de evaluaciones necesitado en cada ejecución para alcanzar el siguiente nivel de precisión, ratio de éxito y rendimiento de éxito.

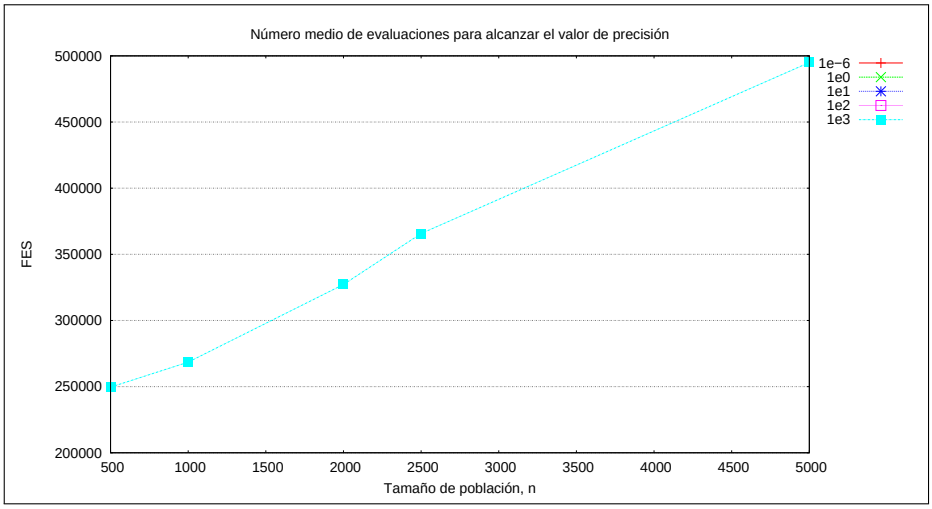

Figura F.42: 50.R.**.0005-Número medio de evaluaciones para alcanzar el valor de precisión

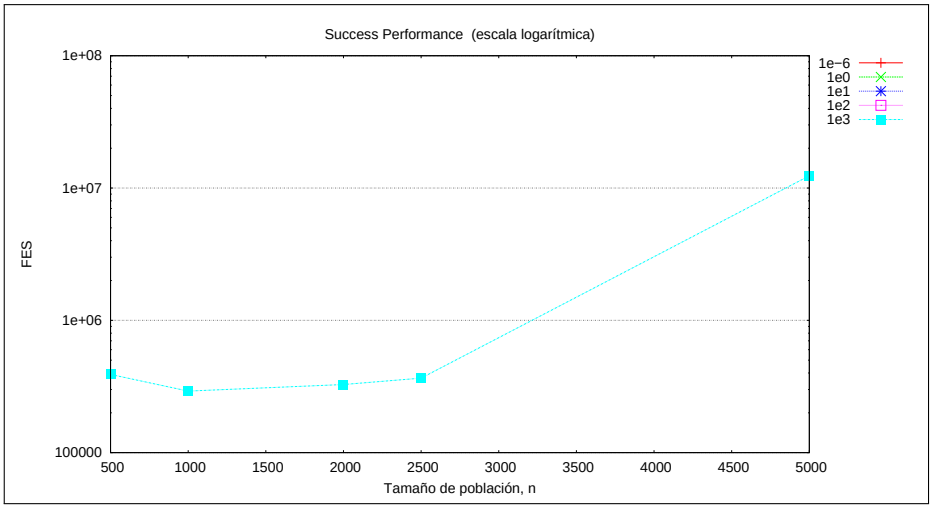

Figura F.43: 50.R.**.0005-Rendimiento de éxito (escala logarítmica)

3. Gráficas de convergencia.

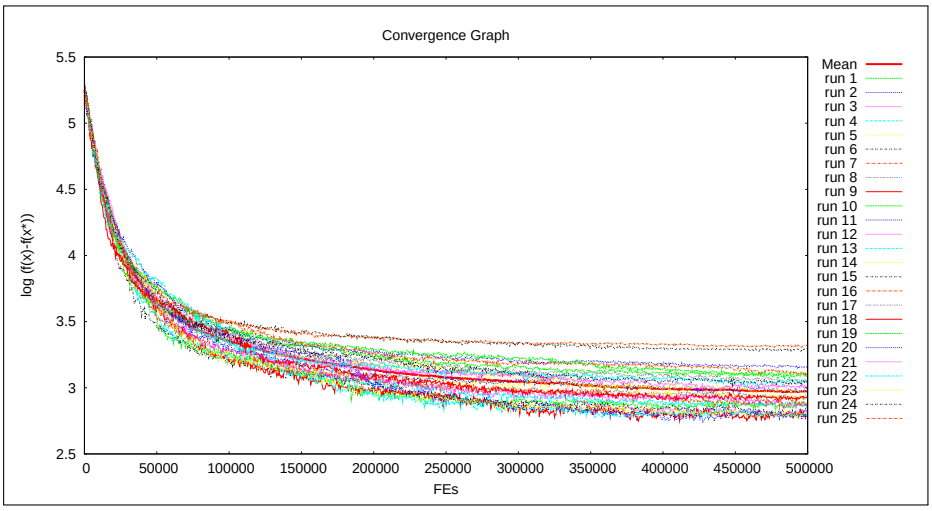

Figura F.44: 50.R.500.0005-Convergencia

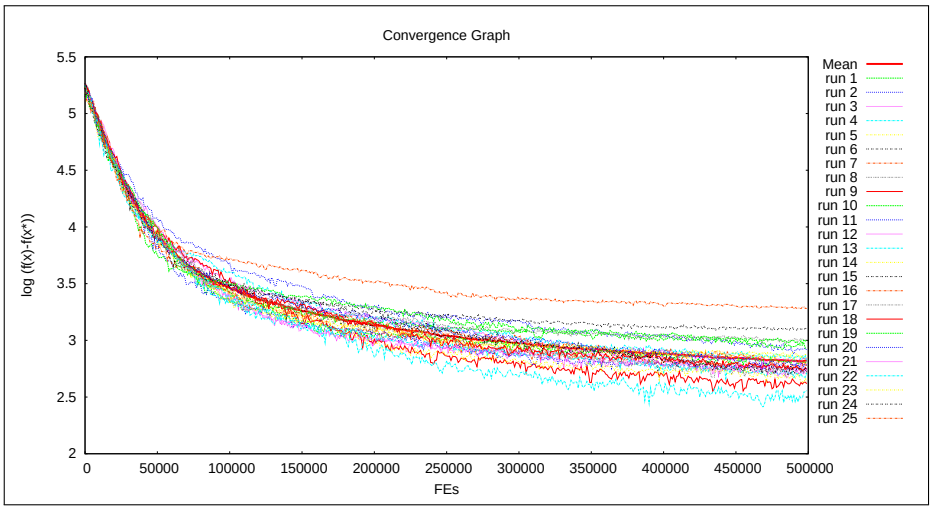

Figura F.45: 50.R.1000.0005-Convergencia

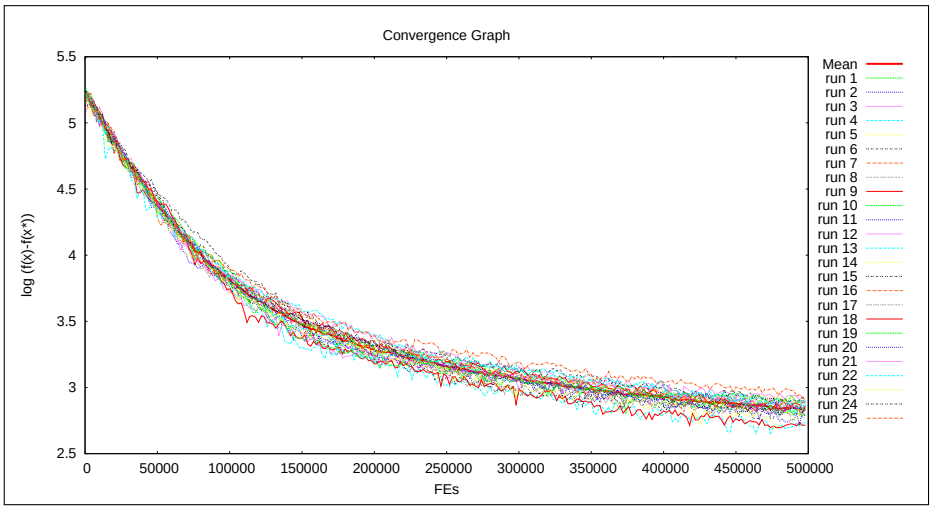

Figura F.46: 50.R.2000.0005-Convergencia

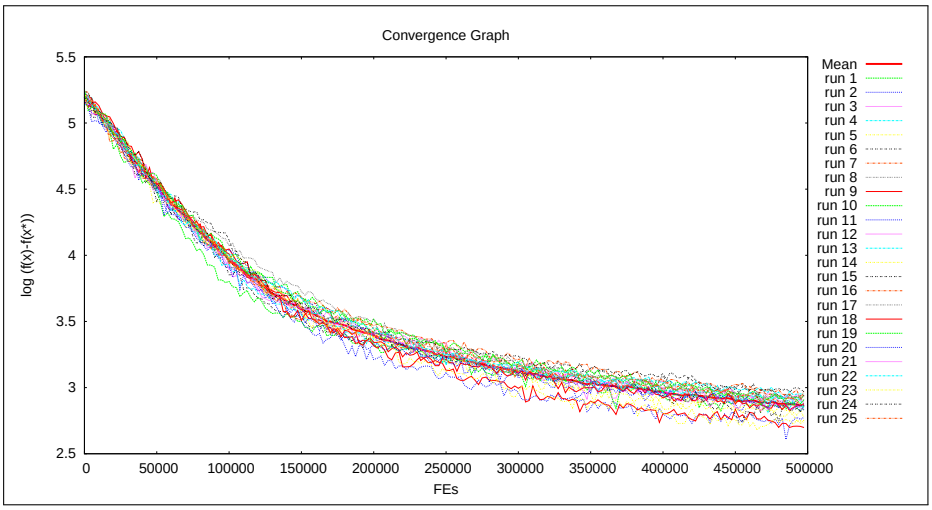

Figura F.47: 50.R.2500.0005-Convergencia

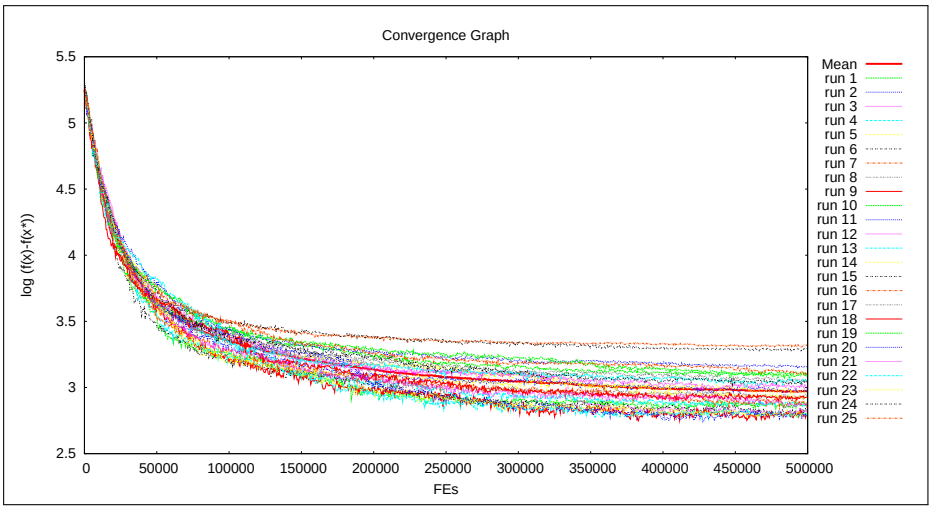

Figura F.48: 50.R.5000.0005-Convergencia

4. Algorithm Complexity.

n	T0	T1	T2’	(T2’-T1)/T0
500	0,18	1,77	494,33	2736,47
1000			498,67	2760,55
2000			502,19	2780,12
2500			500,8	2772,4
5000			506	2801,26

Tabla F.12: 50.R.**.0005-Algorithm Complexity- computing_time_t2

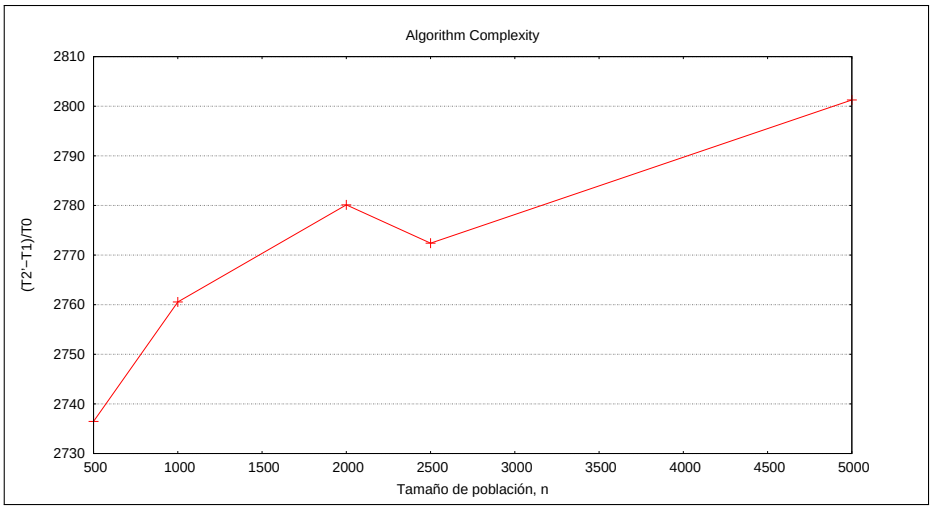

Figura F.49: 50.R.**.0005-Complejidad del algoritmo